Humboldt-Universität zu Berlin
Mathematisch–Naturwissenschaftliche Fakultät II
Institut für Informatik
Lehrstuhl für Künstliche Intelligenz

Ein anderes Modell der Welt

Alternative Methoden zur Lokalisierung Mobiler Roboter

Diplomarbeit

Autor: Heinrich Mellmann
mellmann@informatik.hu-berlin.de

Gutachter: Prof. Dr. Hans-Dieter Burkhard
Prof. Dr. Raúl Rojas

Berlin, den 21. Januar 2015

Many tasks of a mobile robot, e.g., navigation, require the knowledge of the positions of the objects in the surrounding environment. This task is especially challenging for the robots which perception is based on a directed visual system, e.g., a camera with a limited view angle. The incomplete and noisy sensor information leads to the uncertainty in the robots belief of the world. An appropriate model of the world may enable the robot to make plans and to realize complex behavior.

The state of the art modeling methods use often only a small part of the available information. In particular the redundant information remain unused.

In this work we investigate methods to exploit effectively the redundant information in order to get a better model of the world. In the first part we discuss a number of possibilities to use of specific properties of the objects to estimate the parameters of the camera matrix. In the second part we present a constraint based approach for the world modeling.

Für viele Aufgaben (wie etwa Navigation) eines mobilen Roboters ist es notwendig die Lage der Objekte in seiner unmittelbaren Umgebung zu kennen. Eine besondere Herausforderung stellt diese Aufgabe für die Roboter dar, deren Wahrnehmung hauptsächlich auf einem gerichteten, visuellen System basiert, wie etwa eine Kamera mit begrenztem Öffnungswinkel. Die Unvollständigkeit und Ungenauigkeit der Sensorinformation führt zur Unsicherheit in der Vorstellung des Roboters von der Welt. Ein geeignetes Modell von der Welt kann dem Roboter die Fähigkeit zu planen verleihen und ein komplexes Verhalten ermöglichen.

Die aktuellen Verfahren nutzen meistens nur einen geringen Teil der Informationen die dem Roboter zu Verfügung stehen. Insbesondere redundante Informationen bleiben ungenutzt.

In dieser Arbeit stellen wir Verfahren vor, die es zum Ziel haben möglichst gut redundante Informationen bei der Modellierung der Umwelt zu berücksichtigen. Im ersten Teil stellen wir eine Reihe von Möglichkeiten vor wie objektspezifische Eigenschaften verwendet werden können, um die Parameter der Kameramatrix zu korrigieren. Im zweiten Teil wird ein auf Constraints basierender Ansatz für die Modellierung der Welt vorgestellt.

Inhaltsverzeichnis

Ceux qui s'appliquent trop aux petites choses
deviennent ordinairement incapables des grandes.

Jene die sich zu sehr um die kleinen Sachen bemühen
werden für gewöhnlich unfähig für große.

François VI. Duc de La Rochefoucauld (1613 - 1680)
Réflexions ou sentences et maximes morales, XLI

Dieses Manuskript ist eine Zusammenfassung der Ergebnisse aus drei Jahren Forschung innerhalb des RoboCup Projektes. Im Laufe der Untersuchungen ist eine ganze Reihe von Publikationen entstanden die auf internationalen Konferenzen und in Zeitschriften veröffentlicht wurden. Während der gesamten Zeit hatte ich die Gelegenheit mit vielen herausragenden Menschen zu arbeiten. Diese Arbeit wäre ohne ihre Unterstützung undenkbar. Beim Rückblick auf die lange Zeit scheint die Liste der Personen, denen ich danken möchte nahezu endlos.

An erster Stelle möchte ich Prof. Dr. Hans-Dieter Burkhard danken, als der Person, die mich seit meinem zweiten Semester und meinem ersten Prolog-Programm auf meinem Weg durch das Studium begleitet hat und der ich meine Begeisterung für KI und RoboCup verdanke. Von ihm habe ich die Grundlagen und das Verständnis der Künstlichen Intelligenz gelernt.

Ich danke Dr. Daniel Göhring und Matthias Jüngel für die lange, interessante und sehr inspirierende Zusammenarbeit, ich habe viel dabei gelernt. Danke auch für lange Nächte vor den Deadlines und viele Tage und Nächte beim RoboCup.

Ohne die technische Unterstützung der RoboCup-Teams *Nao Team Humboldt*, *Aibo Team Humboldt*, *Humanoid Team Humboldt* und des gesamten *GermanTeam* wäre diese Arbeit unmöglich. Insbesondere möchte ich mich bei Oliver Welter, Thomas Krause, Alexander Borisov, Tobias Hermann, Claas-Norman Ritter, Florian Holzhauer und Yuan Xu für die vielen Stunden am Spielfeld bedanken.

Ich danke auch Dr. Manfred Hild, Michael Spranger und Kateryna Gerasymova für anregende Gespräche und Inspiration. Vielen Dank an Marina Erenberg für das Entfernen von so vielen Fehlern. Danke an Nadja Govedarova für wichtige Hinweise. Danke allen für moralische Unterstützung. Am meisten aber danke ich meinen Eltern, die mich alle diese Jahre unterstützt haben.

Es sind zu viele Namen als das ich alle hier auflisten könnte. Vielen Dank an alle die mich unterstützt haben, es ist euer Werk!

1. Einleitung

Mobile Autonome Robotik ist ein junges und aktuelles Forschungsgebiet. Gleichzeitig ist es auch eines der faszinierendsten Themengebiete der heutigen Zeit. Wir alle träumen von einem vollständig autonomen Roboter[1], der selbständig alle seine Aufgaben erfüllt. Oder besser noch: der seine eigenen Aufgaben formuliert um unser Leben schöner zu gestalten.

Auf das Wort *Intelligenz* bzw. *Kognition* verzichten wir dabei ganz bewusst, denn die Autonomie eines Roboters ist direkt an seine Intelligenz gebunden. Je vielfältiger die Aufgaben eines Roboters sind, die er selbständig lösen soll, desto größer müssen seine *kognitiven* Fähigkeiten sein.

Die Autonomie eines Roboters hängt jedoch stark von seiner Fähigkeit ab, die Umwelt zu erfassen und darin zu navigieren. So kann beispielsweise ein Haushaltsroboter nur dann ein Bier holen, wenn er auch den Kühlschrank findet. Dabei ist die Geschwindigkeit und die Genauigkeit der Wahrnehmung von entscheidender Bedeutung.

Andererseits muss ein autonomer Roboter mobil genug sein um seine Aufgaben erledigen zu können. Je mehr Freiheitsgrade ein Roboter besitzt, desto komplexer können auch seine Bewegungen und sein Verhalten werden. Damit wächst aber auch die Komplexität der Wahrnehmung und Navigation. So kann zum Beispiel ein autonomer Putzroboter in einem Bürogebäude sich immer darauf verlassen, dass der Boden eben ist. Ein Laufroboter, der sich im unwegsamen Gelände zurecht finden soll, kann das nicht. Ganz im Gegenteil muss er damit umgehen können, dass er jederzeit stolpern und hinfallen kann.

Zusätzlich ist Mobilität immer mit eingeschränkten Ressourcen verbunden. Ein stationärer Roboter kann an einen schnellen Computer und Stromnetz angeschlossen sein, während ein frei laufender Roboter immer alles dabei haben muss. So kann ein Roboter von der Größe eines Menschen keinen Computer von der Größe des *Deep Blue*[2] mit sich tragen um seine Aufgaben zu lösen. Das setzt Grenzen für die gesamte Kognition des Roboters.

Wir haben also letztendlich einen klassischen Kompromiss zwischen den verfüg-

[1] Das Wort *Roboter* (tschechisch: *robot*) wird auf die Tschechischen Schriftsteller Josef und Karel Čapek zurückgeführt und wurde zuerst im 1921 erschienenen Drama „R.U.R. – Rossums Universal-Roboter“ verwendet. Sein Ursprung liegt im slawischen Wort *robota* welches mit Arbeit, Fronarbeit oder Zwangsarbeit übersetzt werden kann.

[2] *Deep Blue* Deep Blue war ein von IBM entwickelter Schachcomputer. Ihm gelang es 1997 als erstem Computer der Welt den amtierenden Weltmeister Гарри Каспаров (Gàrri Kaspàrow) im Schach zu schlagen.

baren Ressourcen und der Mobilität. Unter diesem Gesichtspunkt brauchen wir Algorithmen, die die verfügbaren Daten möglichst optimal ausnutzen. Viele Sensorinformationen werden jedoch von den aktuellen Verfahren nicht effizient oder gar nicht genutzt. Insbesondere visuelle Sensordaten enthalten viel mehr Information als normalerweise verwendet wird. Insbesondere bleiben oft viele redundante Daten ungenutzt. Verfahren basierend auf Constraints könnten an dieser Stelle eine interessante Alternative zu klassischen Verfahren darstellen.

1.1. Modellierung der Welt

Unter einem *Weltmodell* eines Roboters wollen wir in dieser Arbeit seine Vorstellung von der eigenen Position und der Lage der Objekte in seiner Umgebung verstehen. Das entspricht im weitesten Sinne auch dem Problem der *Lokalisierung* eines Roboters, daher wollen wir von nun an beide Begriffe synonym verwenden.

Um das Problem besser zu verstehen, versuchen wir zunächst uns eine etwas allgemeinere Sicht darauf zu verschaffen. Als erstes überlegen wir mit welchen Informationen genau wir es zutun haben. Wie bei jedem anderen Problem haben wir gegebene und gesuchte Größen. Für unsere Ziele können wir die gegebene Größen in zwei Gruppen unterteilen: Annahmen über die Welt und Messdaten der Sensoren. Zu den Annahmen über die Welt gehören alle Daten die wir als bekannt voraussetzen, wie etwa die Auflösung der Kamera oder die reale Größe des Balls etc.. Aber auch die Annahmen über die Zusammenhänge gehören dazu, wie zum Beispiel das Modell der Kamera (wie wird ein Punkt auf das Bild projiziert?) oder die kinematische Kette (wie wirkt sich die Rotation an einem Gelenk auf die Position der Kamera aus?). Die Fehlerverteilungen der Sensordaten bei den stochastischen Verfahren gehören ebenfalls zu solchen Annahmen. Zu den Sensormessungen gehören solche Daten wie die Gelenkwinkel, die Kraft an einem Drucksensor, der Farbwert eines Pixels etc.. Die verarbeiteten Daten können wir aber auch als Sensorwerte gewisser abstrakter Sensoren interpretieren. So kann es z.B. einen Sensor geben, der die Größe des Balls im Bild misst.

Gesucht sind bei einem Lokalisierungproblem die räumlichen Relationen zwischen dem Roboter und den Objekten in der Umgebung. Normalerweise ist die Position des Roboters in der Welt oder die Positionen der Objekte relativ zum Roboter von Interesse.

Die Lösung des Lokalisierungsproblems wird üblicherweise als eine Funktion formuliert, die zu gegeben Messdaten die Positionen der Objekte berechnet. Der Algorithmus, der diese Funktion dann berechnet, basiert dabei auf den gemachten Annahmen. Die Annahmen über die Welt reduzieren die Komplexität des Problems, damit gehen aber auch viele Freiheitsgrade und damit verbundene Lösungsmöglichkeiten verloren.

Wenn zum Beispiel die Gelenkwinkel und der Radius des Balls im Bild gemessen wurden, dann kann mit Hilfe der kinematischen Kette die Position der Kamera

relativ zum Roboter bestimmt werden. Damit lässt sich dann mit der bekannten Größe des realen Balls anhand der Größe im Bild die Position des Balls relativ zum Roboter bestimmen. Wenn aber zusätzlich die Position des Ballzentrums im Bild gemessen wurde, dann kann die Position des Balls ebenfalls durch Projektion auf den Boden bestimmt werden. Damit haben zwei verschiedene Messungen dazu verwendet dieselbe Größe zu berechnen. Diese Redundanz könnten wir nun verwenden um Fehler zu minimieren. Z.B. wenn wir annehmen, dass die Messungen der Gelenkwinkel fehlerhaft sind, dann könnten wir sie so korrigieren, dass beide Resultate übereinstimmen. Dadurch kann die Bestimmung der Positionen anderer Objekte verbessert werden.

Wir können das Lokalisierungsproblem allgemeiner als ein Ausgleichs- oder Optimierungsproblem formulieren. Dabei können nicht nur die gesuchten Größen, sondern auch die gemessenen oder vorausgesetzten Größen (wie etwa der Öffnungswinkel der Kamera) als Variablen vorkommen. Andere Annahmen (wie z.B. das Modell der kinematischen Kette) gehen als Gleichungen ein, die erfüllt werden müssen. Die Messungen der Sensoren liefern zusätzliche Abhängigkeiten und Einschränkungen für die variable Größen. Wenn wir z.B. eine Messung eines Gelenkwinkels haben, dann können wir annehmen, dass der echte Winkel in einem bestimmten Bereich um diese Messung liegt.

So können in dem obigen Beispiel die Winkel der Gelenke, die Größe und die Position des Balls im Bild, sowie seine Position relativ zum Roboter als Variablen angesehen werden. Durch das Modell der kinematischen Kette und der Kamera sind dann Zusammenhänge gegeben die alle diese Größen in Relation setzen und dadurch einschränken. Wenn wir nun Messungen für einige dieser Werte haben, so erhalten wir zusätzliche Einschränkungen. Wenn genügend Einschränkungen zur Verfügung stehen, dann können alle Größen als ein Optimum des Problems bestimmt werden.

Welche Größen dabei variabel sein sollen und welche als Wissen vorausgesetzt werden hängt davon ab, ob genügend Information zur Verfügung steht. Ein weiteres Kriterium ist die Sicherheit der Daten: wenn wir die reale Größe des Balls kennen, dann mach es Sinn sie als eine Einschränkung zu verwenden. Ein stark fehlerbehafteter Gelenkwinkel sollte dagegen als Variable modelliert werden, damit er durch andere Informationen korrigiert werden kann.

1.2. Struktur

Diese Arbeit lässt sich in zwei große Bereiche unterteilen:

1. Visuelle Distanzbestimmung anhand von Referenzobjekten
2. Constraint (Deutsch: *Einschränkungen*) basierte Lokalisierung

Im ersten Teil untersuchen wir, wie speziell visuelle Information dazu verwendet werden kann die räumlichen Relationen zu den Objekten zu bestimmen. Üblicherweise

wird die Lage des Roboters anhand seiner kinematischen Kette, Beschleunigungsvektoren und Gyrometer bestimmt. Oft ist aber die Genauigkeit nicht hinreichend. Insbesondere bei Laufrobotern ist es schwierig die Lage des Körpers genau zu bestimmen. Das führt dazu, dass die Lage der Objekte in der Umwelt nicht genau bestimmt werden kann. Hier untersuchen wir exemplarisch wie bekannte Eigenschaften (wie etwa Form oder Erscheinung der Objekte) verwendet werden können um die Lokalisierung zu verbessern.

Um die vorhandenen (insbesondere redundante) Sensorinformationen und Zusammenhänge besser nutzen zu können, können wir die Aufgabe der Lokalisierung des Roboters als ein analytisches Optimierungsproblem auffassen. Allerdings wird der Berechnungsaufwand bereits bei wenigen Variablen sehr groß (vgl. bayessche Filter). Als eine Alternative stellen wir im zweiten Teil der Arbeit einen auf Constraints basierten Modellierungsansatz vor. Dabei wird die Aufgabe der Modellierung der Umwelt als ein Constraint Erfüllungs-Problem formuliert. Die Beschreibung durch Constraints bietet den Vorteil, dass auch hochdimensionale Daten einfach dargestellt und integriert werden können.

1.3. Rahmen der Arbeit

Als Testumgebung/Plattform für unsere Untersuchungen wurde die Umgebung von RoboCup verwendet.

> *RoboCup* ist eine internationale Initiative zur Förderung der Forschung in den Bereichen Künstliche Intelligenz und autonome mobile Systeme (Quelle [31]).

Hier haben die Roboter die Aufgabe in Teams gegeneinander Fußball zu spielen. Das Spielfeld ist ausgestattet mit Toren, Feldlinien und zusätzlichen Landmarken, die die Navigation der Roboter vereinfachen. Alle Objekte auf dem Spielfeld sind farblich markiert, um die visuelle Erkennung zu erleichtern, so sind beispielsweise der Ball orange, das Spielfeld grün und die Linien weiß.

Gespielt wird in unterschiedlichen Ligen die sich insbesondere durch die Art der eingesetzten Roboter unterscheiden. Für uns besonders interessant sind die Aibo und die Nao Roboter aus der *Standard Platform League (SPL)*, sowie die zweibeinigen Roboter der *A-Serie* der Humanoiden-Liga (alle Roboter werden genauer im Anhang B vorgestellt). Alle Plattformen bewegen sich auf Beinen und sind mit einem beweglichen Kopf und einer gerichteten Kamera ausgestattet.

Während eines Fußballspiels müssen die Roboter autonom agieren. Dabei spielt die visuelle Wahrnehmung eine zentrale Rolle. Um in einer solchen dynamischen Umgebung navigieren zu können, ist es von entscheidender Bedeutung die Lage der Objekte in der Umgebung des Roboters genau zu erfassen.

Damit erfüllt RoboCup alle am Anfang beschriebenen „Anforderungen“: die Roboter verfügen über komplexe Bewegungsapparate, besitzen nur eingeschränkte Sensorik und Rechenkapazität und müssen in einer hochdynamischen Umgebung agieren. Das macht RoboCup zu einer hervorragenden Testumgebung für unsere Untersuchungen.

2. Visuelle Distanzbestimmung anhand von Referenzobjekten

Eine große Herausforderung der visuellen Wahrnehmung in der Robotik ist es, die räumlichen Relationen zwischen dem Roboter und den Objekten in seiner Umgebung zu bestimmen. Die Position eines Objektes relativ zum Roboter kann (in Polarkoordinaten) durch die Entfernung und die Richtung zu einem festgelegten Referenzpunkt des Objekts beschrieben werden.

Eine zentrale Frage ist also: wie bestimme ich die relativen Koordinaten eines Punktes, der im Bild gesehen wurde? Banal gesprochen, liegt die wesentliche Schwierigkeit darin, dass die Welt drei Dimensionen besitzt, das Bild allerdings nur zwei. Man stelle sich eine gedachte Linie durch den Ursprung der Kamera und einen festen Punkt im Bild vor. Dann kann man sich einfach überlegen, dass jeder Punkt dieser Linie auf denselben Punkt im Bild abgebildet wird. Wir benötigen also mehr Information um die Position eines Punktes aus seiner Abbildung rekonstruieren zu können.

Es gibt verschiedene Möglichkeiten sich zusätzliche Informationen über die Lage der Punkte im Raum zu beschaffen, sodass eine Rekonstruktion der Position aus den Bilddaten möglich wird. Eine Variante wäre die Verwendung von zwei Kameras, wodurch (wie etwa bei einem Menschen) räumliches Sehen (*Stereosehen*) möglich wird. Ein weiteres Beispiel ist eine Kombination einer Kamera mit einem Laserscanner, der Tiefen-Information zu den Bildern liefert.

Unser Interesse gilt jedoch monokularen Kamerasystemen die keine zusätzliche Sensorik zur Gewinnung der Tiefen-Information besitzen. Um hier die Position gesehener Punkte zu bestimmen müssen zusätzlich Annahmen/Informationen verwendet werden. Insbesondere können Wissen über die Lage der Kamera und bekannte Eigenschaften der Objekte dazu genutzt werden.

Sofern die Lage der Kamera bekannt ist und wir wissen, dass ein gesehener Punkt in der Realität auf dem Boden liegt (z.B. der Fußpunkt eines Objektes), dann kann seine Lage durch eine Projektion des Bildpunktes bestimmt werden. Das liefert eine relativ allgemeine Methode die Lage der Objekte zu bestimmen. Wir bezeichnen diese Vorgehensweise auch als *Peilungsbasierte*-Methode.

Die Lage der Kamera (Kameramatrix) kann wiederum mit Hilfe der kinematischen Kette berechnet werden. Oft ist aber die Genauigkeit nicht hinreichend. Bei den Laufrobotern sind oft die Kontakt-Punkte der Füße mit dem Boden nicht genau bekannt. Außerdem haben die Gelenke oft ein Spiel, was zusammen mit der Ungenauigkeit der Gelenksensoren zu erheblichen Fehlern in der berechneten Lage der

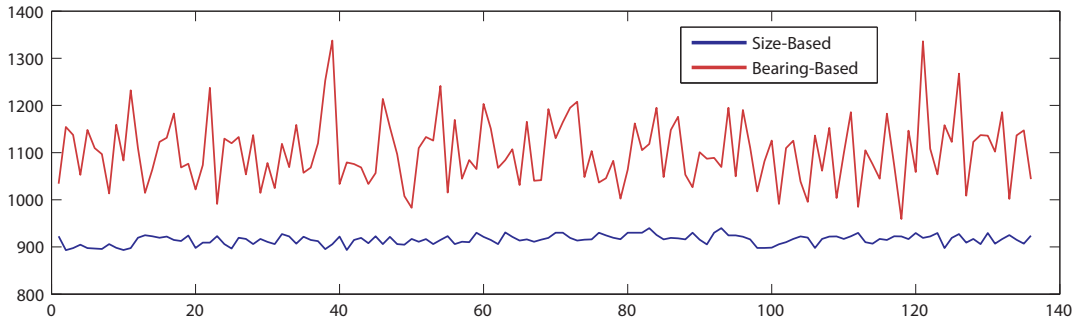


Abbildung 2.1.: *Entfernung zum Ball bestimmt jeweils mit dem Größen- und Peilungsbasierten Verfahren auf einem Aibo ERS-7 Roboter. Der Roboter steht auf der Stelle, Blickrichtung geradeaus, wobei der Ball vor dem Roboter in einer Entfernung von 92cm liegt. Es sind die berechneten Werte über einer Zeit von ca. 6 Sekunden dargestellt.*

Kamera führt. Aber auch bei den Robotern auf Rädern kann es problematisch sein, die Lage der Kamera genau zu bestimmen, insbesondere wenn der Roboter über eine Federung verfügt. Andere Sensoren, wie Beschleunigungssensoren, Drucksensoren an den Füßen und Gyrometer können benutzt werden um die Lage des Roboters genauer zu bestimmen.

Auf diese Weise lässt sich in den meisten Fällen die Richtung, d.h. der horizontale Winkel zu einem Objekt relativ zuverlässig bestimmen. Die bestimmte Entfernung ist dagegen oft sehr fehlerhaft.

Eine der Eigenschaften, die zur Entfernungsbestimmung benutzt werden können ist z.B. die Größe des Objektes. Dafür muss die Größe der Objekte bekannt und darüber hinaus im Bild gut messbar sein, sodass sie mit der realen Größe verglichen werden kann. So ist es zum Beispiel sehr einfach, die Größe des Balls zu bestimmen, die Größe eines Aibo Roboters lässt sich dagegen nicht so einfach definieren, denn abhängig vom Blickwinkel ändert sich die Form und damit auch die Größe des Roboters im Bild. Die Güte der Ergebnisse hängt hier natürlich sehr stark von der Genauigkeit ab, mit der sich die Position und die Größe des Objektes im Bild bestimmen lassen.

In unseren Experimenten hat sich gezeigt, dass die Abschätzung der Entfernung anhand der Größe in der Regel genauer und weniger anfällig gegenüber Fehlern der Bildverarbeitung und in den Sensordaten ist. Die Ursachen dafür werden in den nachfolgenden Abschnitten genauer untersucht.

Die Abbildung 2.1 veranschaulicht als Beispiel die Bestimmung der Entfernung zum Ball mit einem Aibo Roboter. Man sieht sehr deutlich, dass die Ergebnisse des peilungsbasierten Verfahrens sehr stark (teilweise um mehr als 20cm) schwanken und zusätzlich versetzt sind. Wenn der Roboter in Bewegung ist, dann verstärkt sich dieser Effekt.

Analog zur Größe können aber auch andere Eigenschaften der Objekte wie z.B.

die Form dazu verwendet werden die Bestimmung der Entfernung zu verbessern. Diese Information kann auch allgemeiner dazu verwendet werden die Fehler in der berechneten Kameramatrix zu korrigieren. Dadurch kann dann die Entfernung zu anderen gesehenen Objekten verbessert werden.

Allgemeiner formuliert liefert die Kameramatrix den Zusammenhang zwischen der Position des Objektes im Bild und in der Realität (relativ zum Roboter). Auf der anderen Seite existieren abhängig vom Objekt weitere Zusammenhänge zwischen Eigenschaften der Objekte im Bild und in der Realität. Zum Beispiel die Größe, aber auch Ausrichtung, etc.. Eigentlich reicht die Kameramatrix um gesehene Objekte relativ zum Roboter zu lokalisieren. Wenn aber die Kameramatrix nicht genau bestimmt werden kann, so können andere Zusammenhänge dazu benutzt werden, sie zu korrigieren.

Als Motivation für diese Untersuchung dient ursprünglich die Aufgabe der Lokalisierung eines Aibo Roboter in RoboCup. Eine besondere Eigenschaft des Aibo Roboters ist seine geringe Höhe, dadurch werden die Fehler in der Entfernung besonders verstärkt. Außerdem sind die Winkelsensoren der Gelenke auf den Achsen der Motoren montiert. So wird das eigentlich Spiel im Getriebe nicht berücksichtigt.

Die Entfernung zu einer gesehenen Landmarke schränkt mögliche Positionen des Roboters sehr stark ein. Die Genauigkeit der Lokalisierung hängt jedoch direkt von der Genauigkeit der Messung ab. Die meisten Lokalisierungs-Algorithmen die auf dem Aibo eingesetzt werden [18, 20, 23] benutzen nahezu keine Entfernungsinformation aufgrund der großen Ungenauigkeit, sondern stützen sich auf die horizontalen Winkel zu den Objekten. In [30, 29] wird die Entfernung zu den Linienpunkten bei der Lokalisierung verwendet (horizontaler Winkel zu einem Linienpunkt liefert keine Einschränkung für die mögliche Position des Roboters). Die Entfernung zu einem Punkt auf dem Boden kann jedoch beim Aibo nur sehr ungenau bestimmt werden, was dazu führt, dass die Positionsbestimmung instabil werden kann und dazu neigt auszubrechen, wenn keine weitere Information (wie etwa Winkel zu den Landmarken) verfügbar ist. In [30] werden für die Lokalisierung horizontale Winkel zu den Liniensegmenten verwendet um genauere Ergebnisse zu erhalten.

Dieser Abschnitt ist in Teilen in einer Zusammenarbeit mit Matthias Jüngel entstanden. Wesentliche Teile des Abschnitts wurden in zwei Publikationen [22, 26] veröffentlicht.

Dieses Kapitel unterteilt sich in mehrere Abschnitte. Als erstes führen wir einige Grundlagen der visuellen Wahrnehmung, die in späteren Abschnitten benötigt werden, ein. Im zweiten Abschnitt untersuchen wir exemplarisch eine Reihe von Möglichkeiten die externen Parameter der Kamera mit Hilfe von Referenz-Objekten in der Umgebung des Roboters zu bestimmen bzw. zu korrigieren. In dem dritten Abschnitt untersuchen wir die Robustheit dieser Verfahren gegenüber Fehlern. Abschließend stellen wir die Resultate einiger Experimente vor, in denen die Verfahren unter realen Bedingungen auf unterschiedlichen Robotern getestet wurden.

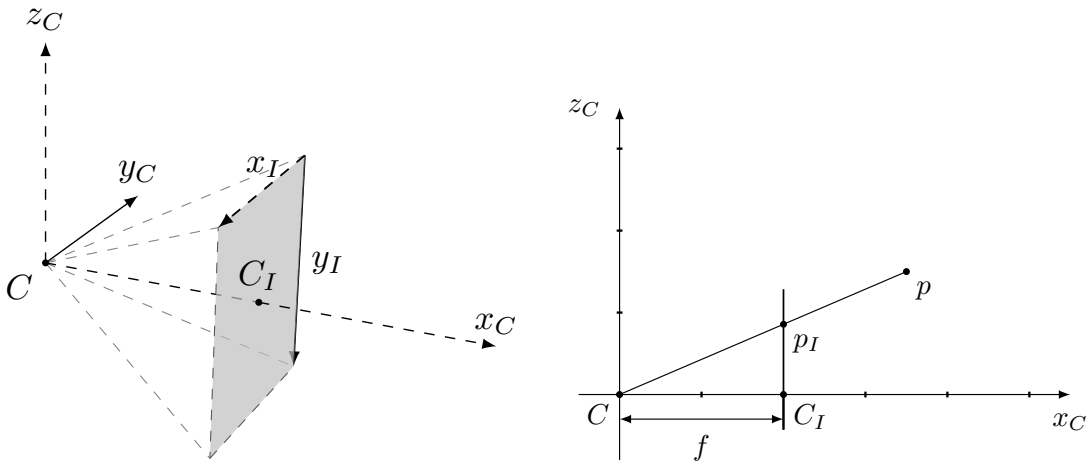


Abbildung 2.2.: *Lochkamera Modell: Die Achsen x_C, y_C und z_C bilden das Koordinatensystem der Kamera. Die Achse x_C bezeichnet man auch als die optische Achse. Der Punkt C markiert das optische Zentrum der Kamera, C_I ist entsprechend das optische Zentrum im Bild. Die Achsen x_I, y_I bilden das Bildkoordinatensystem. Den Abstand f zwischen dem Zentrum C und der Bildebene bezeichnet man als Fokallänge. Die Abbildung der Punkte auf die Bildfläche wird als eine Zentralprojektion am Punkt C modelliert. In der Abbildung rechts stellt der Punkt p_I die Projektion des Punktes p auf die Bildebene dar.*

2.1. Geometrische Grundlagen der Visuellen Wahrnehmung

Mit Hilfe von Bildverarbeitung könne Objekte im Bild (wie etwa der Ball) erkannt werden. Wir wollen von der visuellen Information (z.B. die Position oder die Größe des Balls im Bild) auf die räumliche Relationen schließen (z.B. die Entfernung zum Ball). Dazu brauchen wir eine Vorstellung davon, wie der reale Ball mit seiner Abbildung im Bild zusammenhängt.

In diesem Abschnitt geben wir einen groben Überblick über einige grundlegende Elemente der Geometrie der visuellen Wahrnehmung. Dabei werden insbesondere die Begriffe eingeführt, die im Weiteren verwendet werden.

2.1.1. Projektives Kameramodell

Die Lochkamera ist das einfachste Modell einer Kamera. Dabei nehmen wir an, dass das Licht durch ein unendlich kleines Loch auf die Bildebene fällt. Die Abbildung 2.2 visualisiert ein geometrisches Modell der Lochkamera. Die Abbildung eines Punktes auf die Bildebene wird dabei durch eine Zentralprojektion beschrieben. Die Projektionsebene bezeichnen wir auch als die *Bildebene*.

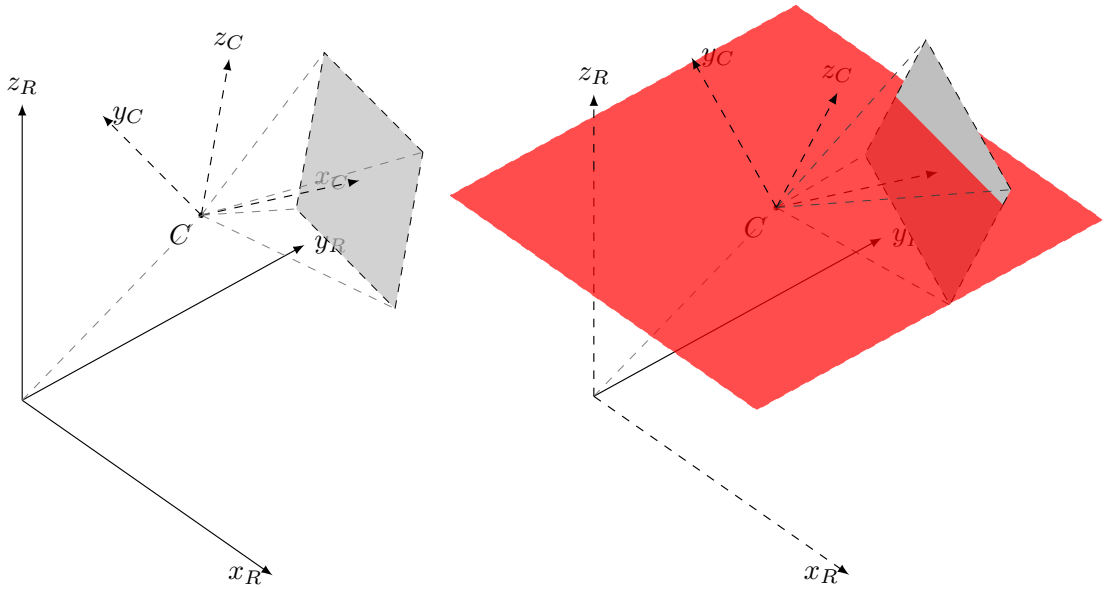


Abbildung 2.3.: *Die Achsen x_R, y_R und z_R bilden das Koordinatensystem des Roboters, die Achsen x_C, y_C und z_C bezeichnen das Kamerakoordinatensystem. De Abbildung rechts visualisiert die Ebene des Horizonts. Diese Ebene verläuft parallel zu der Bodenebene $x_R \times y_R$ durch das optische Zentrum der Kamera C. In manchen Situationen schneidet die Bildfläche die Horizontebene. Dieser Schnitt lässt sich im Bild als eine Linie darstellen.*

Als *optische Achse* bezeichnet man die in Längsrichtung durch die Mitte eines Objektives verlaufende gedachte Linie. In unserem Modell entspricht die optische Achse der x-Achse der Kamerakoordinaten wie in der Abbildung 2.2 dargestellt. Der Abstand zu der Bildebene wird als *Fokallänge* bezeichnet.

Als *äußere*, *externe* oder *extrinsische* Parameter einer Kamera bezeichnet man die Größen die die Lage des optischen Zentrums der Kamera beschreiben. D.h. drei Größen für Position im Raum und drei Größen für die Beschreibung der Rotation. Im Gegensatz dazu beschreiben die *inneren*, *internen* oder *intrinsischen* Parameter einer Kamera ihre internen Werte wie z.B. Auflösung, Fokallänge, etc..

Ein Punkt $p = (x, y, z) \in \mathbb{R}^3$ in den Kamera Koordinaten wird auf den Punkt

$$p_I = \frac{f}{x}(x, y, z) \tag{2.1}$$

auf der Bildebene abgebildet. Die Rotation der Kamera an der Achse y_C bezeichnen wir auch als *Neigung* der Kamera.

2.1.2. Koordinatensysteme

Für uns sind folgende vier Koordinatensysteme von Interesse:

- *Bildkoordinaten* oder *Pixelkoordinaten*
- *Kamerakoordinaten*
- *Roboterkoordinaten*
- *Weltkoordinaten*

Die Bildkoordinaten werden in Zusammenhang mit den Kamerakoordinaten in der Abbildung 2.2 (links) veranschaulicht. Die Abbildung 2.3 (links) visualisiert das Koordinatensystem des Roboters und seiner Kamera.

Wir bezeichnen die xy-Ebene der Roboterkoordinaten als die *Bodenebene*. Die Position der Objekte wird relativ zu den Koordinaten des Roboters beschrieben.

Wir gehen von einer flachen Welt aus, d.h. der Boden ist immer eben. Die Position des Roboters $(x, y, \alpha) \in \mathbb{R}^2 \times [-\pi, \pi]$ relativ zu den Weltkoordinaten wird durch seine Position und Ausrichtung in der xy-Ebene beschrieben. Das Koordinatensystem des Roboters entspricht bis auf die Transformation entsprechend seiner Position den Weltkoordinaten. Insbesondere ist die z-Achse der Roboterkoordinaten parallel zu der z-Achse der Weltkoordinaten.

2.1.3. Horizont

Sei die das optische Zentrum der Kamera O im Roboter Koordinaten gegeben, dann bezeichnen wir die Ebene H definiert durch

$$H : O + t \cdot (1, 0, 0) + s \cdot (0, 1, 0), \quad s, t \in \mathbb{R} \tag{2.2}$$

als *Horizont*. Intuitiv ist das die Ebene, die durch das optische Zentrum der Kamera verläuft und parallel zum Boden ist. Für bestimmte Orientierungen der Kamera schneidet H die Bildebene. Die Schnittkante im Bild bezeichnen wir als die *Linie des Horizonts* im Bild. Die Vertikale Verschiebung dieser Linie im Bild entspricht dem Neigungswinkel der Kamera, deren Anstieg der Rotation der Kamera um die optische Achse. Die Rotation an der z-Achse hat offensichtlich keinen Einfluss auf den Schnitt der Bildebene mit dem Horizont. Intuition: wenn wir die Linie des Horizonts im Bild sehen würden, dann könnten die Neigung und die Rotation der Kamera direkt bestimmt werden. Im Abschnitt 2.2.3 greifen wir diese Idee auf.

2.1.4. Kameramatrix und Kameratransformation

Wir betrachten die Kamera immer relativ zu den Koordinaten des Roboters.

Als *Kameramatrix* oder (Kamera-) *Projektionsmatrix* bezeichnet man im allgemeinen eine 3×4 Matrix, die die Projektion der 3D Punkte in den Roboterkoordinaten

auf die 2D Punkte im Bild beschreibt. Diese Matrix enthält die internen als auch die externen Parameter der Kamera.

Als *Kameratransformation* bezeichnen wir die Koordinatentransformation von dem Kamera-Koordinatensystem in das Koordinatensystem des Roboters.

Die Lage der Kamera relativ zum Roboter kann durch sechs Parameter beschrieben werden: einem Vektor $(x, y, z)^T \in \mathbb{R}^3$, der die Position des optischen Zentrums der Kamera beschreibt und den Winkeln α, β und γ, die die Orientierung der Kamera beschreiben.

Insgesamt können wir die Transformation mit Hilfe einer Rotation und einer Translation durchführen. Wir beschreiben also die Kamera-Transformation durch eine Rotationsmatrix R und einen Translationsvektor T. Die Rotationsmatrix setzt sich wie folgt zusammen:

$$R(\alpha, \beta, \gamma) = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma), \tag{2.3}$$

wobei R_x, R_y und R_z die Rotationsmatrizen an den entsprechenden Achsen bezeichnen, d.h.

$$R_z(\alpha) := \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

$$R_y(\alpha) := \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \tag{2.5}$$

$$R_x(\alpha) := \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{2.6}$$

Wir schreiben die Kameratransformation in der homogenen Form als Funktion ihrer Parameter

$$M = M(\alpha, \beta, \gamma, \omega) := \begin{pmatrix} R(\alpha, \beta, \gamma) & -\omega \\ \vec{0} & 1 \end{pmatrix} \tag{2.7}$$

wobei $\omega \in \mathbb{R}^3$ der Translationsvektor der Kamera ist. Zu einem Punkt $x \in \mathbb{R}^3$ in Roboterkoordinaten können wir seine Repräsentation in den Kamerakoordinaten berechnen durch

$$\begin{pmatrix} \tilde{x} \\ 1 \end{pmatrix} = M \cdot \begin{pmatrix} x \\ 1 \end{pmatrix}. \tag{2.8}$$

2.1.5. Projektion von Bildpunkten auf die Bodenebene

Im Normalfall wollen wir nachdem ein Objekt im Bild erkannt wurde seine Position relativ zum Roboter bestimmen. Diese Projektion stell gewissermaßen eine Inverse der Kameramatrix dar.

Sei zunächst die Kameratransformation

$$M = \begin{pmatrix} R & -\omega \\ \vec{0} & 1 \end{pmatrix} \tag{2.9}$$

gegeben. Ist nun ein Punkt $p = (x, y)$ im Bild gegeben, dann lauten seine Koordinaten im Koordinatensystem der Kamera

$$\tilde{p} = \begin{pmatrix} f \\ (x - c_x) \cdot \delta_x \\ (c_y - y) \cdot \delta_y \end{pmatrix}, \tag{2.10}$$

wobei f die *Fokallänge* der Kamera ist. Der Punkt (c_x, c_y) bezeichnet das optische Zentrum in Pixelkoordinaten und δ_x bzw. δ_y die Breite bzw. die Höhe eines Pixels. Um die Projektion des Punktes p auf den Boden zu berechnen, führen wir zunächst die Transformation in die Weltkoordinaten durch:

$$P := R \cdot \tilde{p} + \omega. \tag{2.11}$$

Nun berachten wir die Gerade durch das optische Zentrum der Kamera ω und den Punkt P und bestimmen deren Schnittpunkt mit der Bodenebene, d.h. wir suchen ein $t \in \mathbb{R}$ so, dass gilt

$$\omega + t \cdot (P - \omega) = \omega + t \cdot R \cdot \tilde{p} = (x, y, 0)^T \tag{2.12}$$

mit irgendwelchen Koordinaten $x, y \in \mathbb{R}$. Wichtig ist dabei, dass die Höhe gleich 0 ist. Somit haben wir also die Gleichung

$$\omega_z + t \cdot (P_z - \omega_z) = 0 \tag{2.13}$$

zu lösen. Wir erhalten also den Projizierten Punkt $\tilde{P}$ durch

$$\tilde{P} = \omega - \frac{\omega_z}{P_z - \omega_z} \cdot R \cdot (f, x, y)^T. \tag{2.14}$$

2.2. Verfahren

Einige objektspezifischer Eigenschaften (wie etwa die Größe des Balls) können dazu benutzt werden räumliche Relationen zwischen dem Roboter und den Objekten zu bestimmen. Die Kameramatrix bietet ebenfalls eine Möglichkeit die Relationen zu anderen Objekten in der Umgebung des Roboters zu bestimmen, insbesondere auch zu solchen die keine einfach zu definierenden Eigenschaften besitzen die das erlauben würden (wie zum Beispiel ein anderer Roboter, dessen Form und Größe sich permanent ändern kann). Im Folgenden stellen wir exemplarisch eine Reihe von Möglichkeiten vor wie Objekte als Referenzen verwendet werden können um die

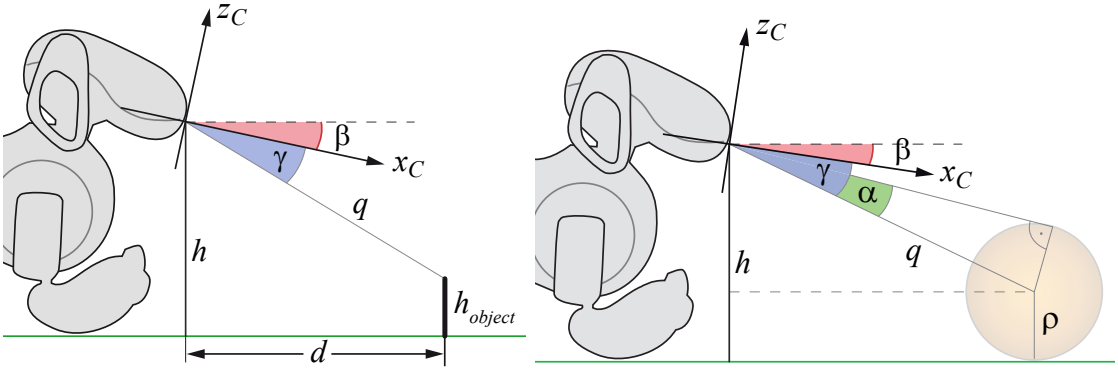


Abbildung 2.4.: *Zusammenhang zwischen dem Neigungswinkel der Kamera und dem Ball als Referenzobjekt. Die linke Abbildung illustriert eine Skizze der peilungsbasierten Entfernungsbestimmung. β ist der Neigungswinkel der Kamera, d.h. der Rotationswinkel der Kamera an der y_R Achse. Man beachte, dass der Winkel γ in diesem Beispiel negativ ist. Der Peilungswinkel zum Objekt ergibt sich durch $\frac{\pi}{2} - \beta + \gamma$. d bezeichnet die Entfernung zum Objekt und q die Entfernung zwischen dem Objekt und dem optischen Zentrum der Kamera.*

Kameramatrix zu korrigiert, wodurch insbesondere die Bestimmung der Entfernung verbessert werden kann.

Essentiell für diese peilungsbasierte Entfernungsbestimmung ist der Neigungswinkel der Kamera. Daher konzentrieren sich die folgenden Betrachtungen im Wesentlichen auf dessen Bestimmung bzw. Korrektur. Wenn wir davon ausgehen, dass die Kamera nicht um die optische Achse rotiert ist, dann können wir unsere Betrachtungen auf ein zweidimensionales Modell beschränken, wie in der Abbildung 2.4 (links) veranschaulicht.

Wie bereits in der Einleitung angesprochen, lässt sich in einigen Fällen die Größe eines Objektes (wie etwa von einem Ball) dazu verwenden die Entfernung zum Objekt abzuschätzen, womit dann der Neigungswinkel der Kamera bestimmt werden kann. Objekte höher als das optische Zentrum der Kamera können dazu verwendet werden den Horizont abzuschätzen und so die Neigung der Kamera zu bestimmen. Die letzte Eigenschaft die wir betrachten werden, ist die Form auf dem Boden. Wenn diese bekannt ist (z.B. Ecken von Feldlinien, oder das Muster vom Teppich), dann liefern Verzerrungen in der Projektion der Form Hinweise auf Fehler in der Kameramatrix.

2.2.1. Objekte bekannter Größe

Die Größe eines Objektes wurde bereits mehrfach als Beispiel für eine Eigenschaft, die für die Korrektur der Kameramatrix verwendet werden kann, aufgeführt. Etwas allgemeiner Formuliert lässt sich mit Hilfe der Größe oft die direkte Entfernung

zwischen dem optischen Zentrum der Kamera und dem Objekt bestimmen. In der Abbildung 2.4 wird diese Entfernung mit q bezeichnet. Aus der Abbildung 2.4 (links) lässt sich nun folgender Zusammenhang zwischen dem Neigungswinkel β und der Entfernung q ableiten:

$$\cos(\frac{\pi}{2} - \beta + \gamma) = \frac{h - h_{object}}{q} \tag{2.15}$$

Der Winkel γ beschreibt den gemessenen vertikalen Winkel zwischen der optischen Achse und dem Objekt im Bild. Wenn wir nun davon ausgehen, dass die Entfernung q zu einem Objekt bekannt ist, so kann also der Neigungswinkel β wie folgt berechnet werden

$$\beta = \arccos\left(\frac{h - h_{object}}{q}\right) - \gamma + \frac{\pi}{2} \tag{2.16}$$

Die Höhe der Kamera h sowie des Objektes h_{object} setzen wir zunächst als bekannt voraus. In späteren Abschnitten werden wir darauf eingehen, wie man den Einfluss von h abschwächen kann.

Die Abbildung 2.4 (rechts) veranschaulicht als Beispiel einen Zusammenhang zwischen dem Neigungswinkel der Kamera und dem Ball als Referenzobjekt[1]. Aus dem gemessenen Winkel α für die Größe des Balls können wir die Entfernung q einfach berechnen durch:

$$\sin(\alpha) = \frac{\rho}{q} \quad \Leftrightarrow \quad q = \frac{\rho}{\sin(\alpha)}. \tag{2.17}$$

In diesem Beispiel entspricht die Höhe des Objekts h_{object} dem Radius des Balls ρ. Wenn wir nun ρ und q in die Formel 2.16 einsetzen, erhalten wir

$$\beta = \arccos\left(\frac{h - \rho}{\rho} \cdot \sin(\alpha)\right) - \gamma + \frac{\pi}{2}. \tag{2.18}$$

Nach dieser Formel können wir nun den Neigungswinkel mit Hilfe der Größe des Balls berechnen. Die einzige Stelle an der dabei zusätzliche Sensorinformationen einfließen ist die Höhe h der Kamera. Wie wir aber später bei der Fehlerbetrachtung sehen werden ist diese Methode weitaus stabiler gegenüber von Fehlern, als die Berechnung mittels der kinematischen Kette.

2.2.2. Objekte bekannter Form auf dem Boden

Das von der Kamera aufgenommene Bild kann mit Hilfe der Kameramatrix auf die Bodenebene projiziert werden, wie im Abschnitt 2.1.5 beschrieben. Wenn die Kameramatrix die tatsächliche Lage der Kamera beschreibt, so erscheinen alle projizierten

[1] Der Vorteil des Balls als Referenzobjekt ist, dass einerseits seine Größe einfach bestimmt werden kann, da er aus allen Richtungen gleich aussieht, und andererseits dass er auf vielen Bildern - als zentrales Objekt des Spiels - zu sehen ist

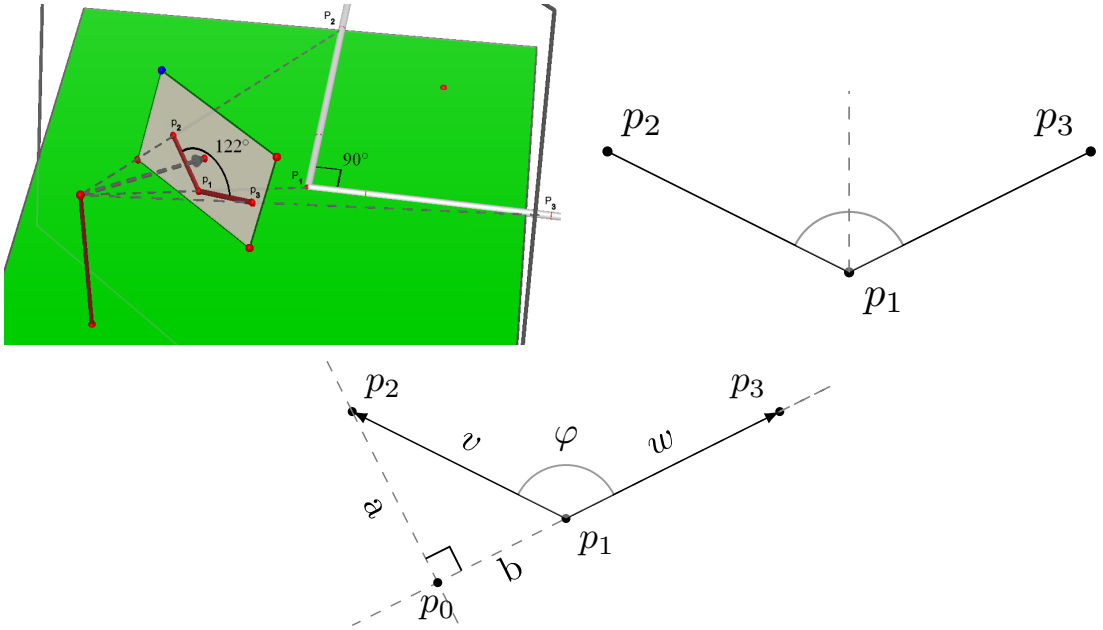


Abbildung 2.5.: *Die Abbildung oben links illustriert den geometrischen Zusammenhang zwischen einer Ecke und ihrer Abbildung. Oben rechts wird die Nummerierung der Punkte dargestellt. Die Punktierte Linie ist die Bisectrix des Winkels. Die Untere Abbildung illustriert die Geometrie der Ecke für die Bestimmung des Winkels.*

Strukturen auf dem Boden (wie etwa die Feldlinien) unverzerrt. Treten dagegen Verzerrungen auf (z.B. die Ecken der Feldlinien sind nicht mehr rechtwinklig), so ist es ein Hinweis darauf, dass die verwendete Kameramatrix nicht korrekt ist. Wir können also versuchen die Kameramatrix so zu korrigieren, dass keine Verzerrungen in der Projektion mehr auftreten. Für diese Art der Korrektur bieten sich im RoboCup die Ecken der Feldlinien an. Als Maß für die Verzerrung benutzen wir den Winkel der projizierten Ecke.

Formal lässt sich diese Idee wie folgt beschreiben: seien p_1, p_2 und p_3 die definierenden Punkte einer Ecke im Bild, wobei p_1 den Eckpunkt bezeichnet. Wir projizieren[2] nun diese Punkte mit Hilfe der Kameramatrix $M(\beta)$ auf die Ebene, wobei β der Neigungswinkel ist. Die resultierenden Punkte bezeichnen wir mit $P_i(\beta) \in \mathbb{R}^3$. Seien weiter

$$v = v(\beta) := P_2(\beta) - P_1(\beta) \tag{2.19}$$
$$w = w(\beta) := P_3(\beta) - P_1(\beta) \tag{2.20}$$

die Vektoren vom Eckpunkt zu den äußeren Punkten, wie in der Abbildung 2.5

[2] wie im Abschnitt 2.1.5 beschrieben

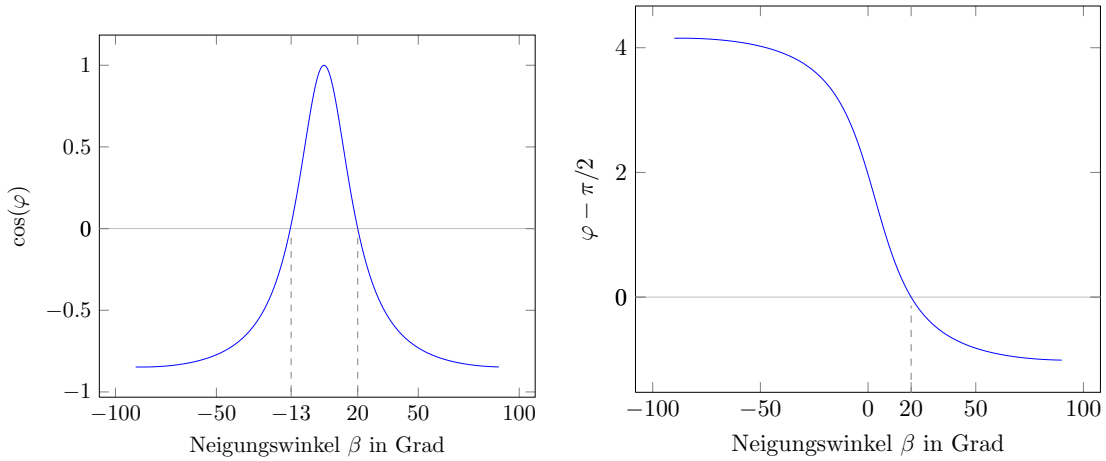


Abbildung 2.6.: *Beispiel für den Winkel einer projizierten Ecke als Funktion des Neigungswinkels der Kamera. In beiden Fällen markieren die Nullstellen die Neigungswinkel bei denen die projizierte Ecke rechtwinklig ist. Der korrekte Neigungswinkel betrug* 20°. *Bei der linken Funktion wird die Ausrichtung der Ecke nicht berücksichtigt, daher existieren zwei mögliche Lösungen. Die negative Lösung projiziert die Ecke spiegelsymmetrisch hinter den Roboter. Bei der rechten Funktion wird die Ausrichtung der Ecke einbezogen, wodurch die Lösung eindeutig wird.*

veranschaulicht. Dann gilt für den Winkel $\varphi = \varphi(\beta)$, der durch diese Punkte definiert wird:

$$\cos\varphi = \frac{\langle v, w\rangle}{||v|| \cdot ||w||}. \tag{2.21}$$

Die Abbildung 2.6 (links) visualisiert den Cosinus $\cos\varphi(\beta)$ des Winkels $\varphi(\beta)$ der Ecke als Funktion des Neigungswinkels der Kamera β. Wir wissen jedoch, dass die Ecken der Feldlinien immer rechtwinklig sind, d.h. es gilt $\varphi = \frac{\pi}{2}$ und somit $\cos\varphi(\beta) = 0$. Damit lässt sich als der korrekte Neigungswinkel β als Nullstelle bestimmen. Mit dieser Bedingung lässt sich die obige Gleichung für β vereinfachen zu:

$$\langle v, w\rangle = 0. \tag{2.22}$$

Im Allgemeinen hat diese Gleichung unendlich viele Lösungen. Im Anwendungsfall des Aibo Roboters liegt allerdings nur eine Lösung im zulässigen Bereich. Wie man leicht in der Abbildung 2.6 (links) sehen kann, existieren zwei Nullstellen, die als Neigungswinkel von der Kamera angenommen werden könnten. Allerdings befindet sich eine Nullstelle im negativen Bereich, d.h. der Roboter würde in diesem Fall nach oben schauen. Bei diesem Neigungswinkel hat die projizierte Ecke zwar einen rechten Winkel, befindet sich aber hinter dem Roboter, was aus physikalischer Sicht keinen Sinn macht. Damit ist für uns nur die zweite Lösung interessant.

Die Tatsache, dass in diesem Fall theoretisch zwei Lösungen möglich sind ist die

Folge davon, dass der Cosinus eine symmetrische Funktion ist. Auf diese Weise berücksichtigen wir nur den absoluten Winkel der projizierten Ecke, aber nicht die Ausrichtung. Um die Ausrichtung zu berücksichtigen führen wir für die Punkte p_i eine Ordnung ein, wie in der Abbildung 2.5(rechts oben) veranschaulicht. Der Eckpunkt hat immer die Bezeichnung p_1, während die Bezeichnung anderer Punkte bezüglich der Bisektrix verteilt werden: der Punkt in der mathematisch positiver Rotationsrichtung von der Bisektrix erhält die Bezeichnung p_2 und der andere p_3. Nun können wir wie in der 2.5(unter) illustriert folgende Hilfsgrößen bestimmen:

$$b = \frac{\langle v, w \rangle}{\|w\|} \tag{2.23}$$

$$a = \frac{\langle v, R_z(\pi/2) \cdot w \rangle}{\|w\|} \tag{2.24}$$

Geometrisch gesehen „zerlegen“ wir den Vektor v in die zum Vektor w parallele Komponente b und die zu w orthogonale a. Es gilt also

$$v = b \cdot \frac{w}{\|w\|} + a \cdot \frac{R_z(\pi/2) \cdot w}{\|w\|}. \tag{2.25}$$

Damit lässt sich nun der Winkel der Ecke bestimmen durch

$$\varphi = \pi - \text{atan2}(a, b). \tag{2.26}$$

Dieser Zusammenhang gilt für jede Art von Ecken, dabei können a und b auch negativ werden. Daher ist die Verwendung von atan2 an dieser Stelle sehr wichtig. Die Abbildung 2.6 (rechts) visualisiert die Funktion

$$\varphi(\beta) = \pi - \text{atan2}(a, b) - \frac{\pi}{2} = \frac{\pi}{2} - \text{atan2}(a, b). \tag{2.27}$$

Der korrekte Neigungswinkel lässt sich hier als einzig mögliche Nullstelle bestimmen.

Diese Lösung lässt sich sehr einfach mit Hilfe von Standardverfahren, wie zum Beispiel dem Newton-Verfahren, finden. Das Newton-Verfahren wird im Anhang A.1 näher erläutert.

Besonders gut funktioniert dieses Verfahren, wenn die verwendete Ecke von innen oder von außen betrachtet wird. Wenn allerdings der Roboter auf einer der Linien der Ecke steht, so wird der Winkel nicht mehr durch einen falschen Neigungswinkel verzerrt und das Verfahren schlägt fehl.

Ein weiteres Detail, das beachtet werden sollte ist, dass für die Punkte im Bild, die auf dem Horizont liegen, keine Projektion auf die Bodenebene möglich ist. In solchen Fällen weisen die oben vorgestellten Funktionen eine Singularität auf, was negative Auswirkungen auf die Berechnung der Nullstellen haben kann.

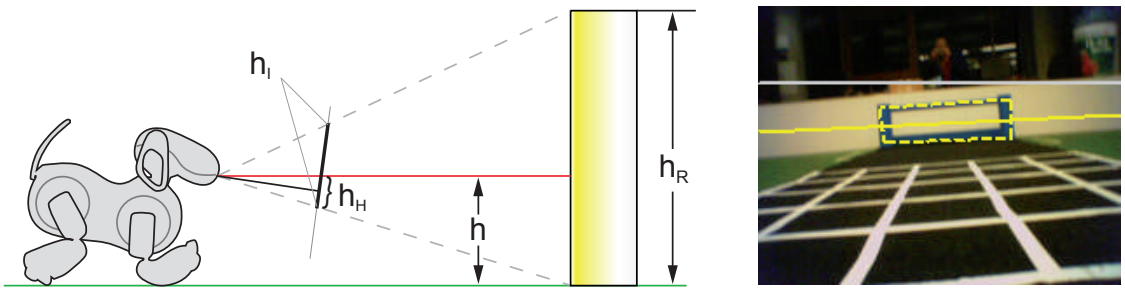


Abbildung 2.7.: *(links) Die Landmarke wird auf die Bildfläche projiziert. Das Wissen über die Höhe der Landmarke kann dazu benutzt werden um die Höhe des Horizonts im Bild zu bestimmen. (rechts) Ein von dem Roboter aufgenommenes Bild. Eingezeichnet sind das erkannte Tor und der jeweils mit Hilfe der Kameramatrix und mit Hilfe des Tores berechnete Horizont.*

2.2.3. Objekte höher als der Horizont

Eine weitere Methode zur Korrektur der Kameramatrix ergibt sich durch die Betrachtung des Horizonts. Oft lässt sich der Horizont an den Objekten die höher sind als der Brennpunkt der Kamera ganz einfach ablesen. Zunächst konzentrieren wir uns nur auf den Neigungswinkel der Kamera, die Korrektur des Rotationswinkels wird im Abschnitt 2.2.4 behandelt. Im Folgenden werden zwei Möglichkeiten vorgestellt, wie die Höhe des Horizonts im Bild bzw. der Neigungswinkel der Kamera ermittelt werden kann.

Schätzung des Horizonts mit Strahlensatz

Eine einfache Methode für die Bestimmung der Höhe des Horizonts im Bild liefert der Strahlensatz. Wir betrachten folgende Situation: der Roboter sieht eine Landmarke deren reale Höhe h_R und die Höhe im Bild h_I bekannt sind. Dann kann die Höhe des Horizonts im Bild einfach auf der Landmarke abgetragen werden, und zwar bei der Höhe

$$h_H \approx \frac{h \cdot h_I}{h_R} \tag{2.28}$$

wie in der Abbildung 2.7 veranschaulicht. Diese Formel ergibt sich direkt mit Hilfe des Strahlensatzes, wenn wir für den Neigungswinkel der Kamera $\beta = 0$ annehmen. In diesem Fall liegt die optische Achse der Kamera genau in der Ebene des Horizonts, d.h. die Bildebene steht orthogonal zum Horizont. Ist β ungleich Null so ergeben sich Fehler, die aber bei einem geringen vertikalen Öffnungswinkel (z.B. 44° bei Aibo Roboter) nicht gravierend sind. Diese Fehler werden im Abschnitt 2.3.2 genauer untersucht.

Nach der Definition des Horizonts liegt das Zentrum des Bildes genau dann auf

der Linie des Horizonts, wenn der Neigungswinkel der Kamera genau $\beta = 0$ beträgt. Auf diese Weise können wir den Neigungswinkel direkt am Horizont ablesen.

Wir können den Winkel β aber auch direkt ausrechnen. Aus der Abbildung 2.8 lässt sich folgende Formel für die Höhe des Horizonts im Bild (gemessen an der Landmarke) herleiten

$$h_H = f \cdot (\tan(\beta) + \tan(\delta)) \tag{2.29}$$

und für die Höhe der gesamten Landmarke im Bild gilt

$$h_I = f \cdot (\tan(\delta) + \tan(\gamma)) \,. \tag{2.30}$$

Lösen wir nun die erste Formel nach β auf, so erhalten wir nach dem Einsetzen der Strahlensatz-Näherung für h_H

$$\beta = \arctan\left(\frac{h \cdot h_I}{f \cdot h_R} - \tan(\delta)\right). \tag{2.31}$$

Nun ersetzen wir h_I und erhalten

$$\beta = \arctan\left(\frac{h - H}{H}\tan(\delta) - \frac{h}{H}\tan(\gamma)\right). \tag{2.32}$$

Diese Formel eignet sich besser für die analytischen Betrachtungen (so etwa für die Untersuchung der Fehler im Abschnitt 2.3.2).

Implizite Berechnung des Neigungswinkels

Eine andere Möglichkeit um den Neigungswinkel der Kamera anhand einer Landmarke zu bestimmen ergibt sich aus der Betrachtung der Zusammenhänge zwischen dem Neigungswinkel und der Projektion der Landmarke auf die Bildfläche wie in Abbildung 2.8 veranschaulicht. In diesem Fall leiten wir eine direkte Abhängigkeit des gesuchten Winkels β von den gemessenen Winkeln γ und δ her. Die Höhe des Horizonts taucht bei dieser Betrachtung explizit nicht auf.

Aus der Skizze in der Abbildung 2.8 lassen sich sofort folgende Zusammenhänge ablesen

$$\tan(\beta + \delta) = \frac{h}{d} \tag{2.33}$$

$$\tan(\gamma - \beta) = \frac{h_R - h}{d} \tag{2.34}$$

Interessant für uns ist dabei der Winkel β. Lösen wir die Gleichung 2.33 nach d auf und setzen das Ergebnis in 2.34 ein, so erhalten wir

$$\tan(\gamma - \beta) = \frac{h_R - h}{h} \cdot \tan(\beta + \delta) \tag{2.35}$$

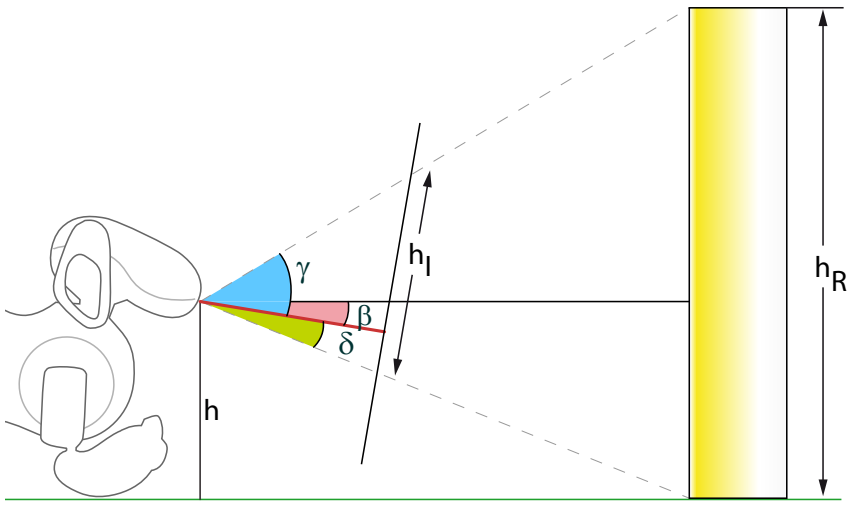


Abbildung 2.8.: *Der geometrische Zusammenhang zwischen dem Neigungswinkel der Kamera β, der Höhe der Landmarke im Bild h_I und der vertikalen Position der Landmarke im Bild wird illustriert. Die Winkel γ und δ lassen sich aus dem Bild ablesen, diese Winkel entsprechen den Anteilen der Landmarke die über bzw. unter dem Zentrum des Bildes liegen. Das optische Zentrum des Bildes wird durch die* ***rote*** *Linie visualisiert. Aus dieser Skizze lassen sich leicht Zusammenhänge ableiten mit deren Hilfe der Neigungswinkel β bestimmt werden kann.*

Nun definieren wir die Funktion

$$f(\beta) := \left[\gamma - \beta - \arctan\left(\frac{h_R - h}{h} \cdot \tan\left(\beta + \delta\right)\right)\right]^2 \tag{2.36}$$

Alle Parameter, die in der Funktion f vorkommen, lassen sich aus dem Bild extrahieren. Damit lässt sich also der korrekte Winkel β als Nullstelle von f bestimmen.

Die Nullstelle kann zum Beispiel mit dem Newton-Verfahren schnell gefunden werden. Als Startwert kann man dabei den aus den Gelenken ermittelten Neigungswinkel verwenden. Alternativ dazu kann der Startwert mit dem Verfahren aus dem vorherigen Abschnitt 2.2.3 berechnet werden.

Zu bemerken ist noch, dass viele verschiedene Zusammenhänge hergeleitet werden können, die wieder auf das gleiche Problem führen, wie zum Beispiel

$$g(\beta) := \left(h \cdot \tan\left(\gamma - \beta\right) - \left(h_R - h\right) \cdot \tan\left(\beta + \delta\right)\right)^2 \tag{2.37}$$

Die Funktion f wurde ausgewählt, da sie gegenüber anderen ein besseres Verhalten bezüglich der Nullstellen-Suche aufweist. Die Abbildung 2.9 veranschaulicht die Funktionen f und g.

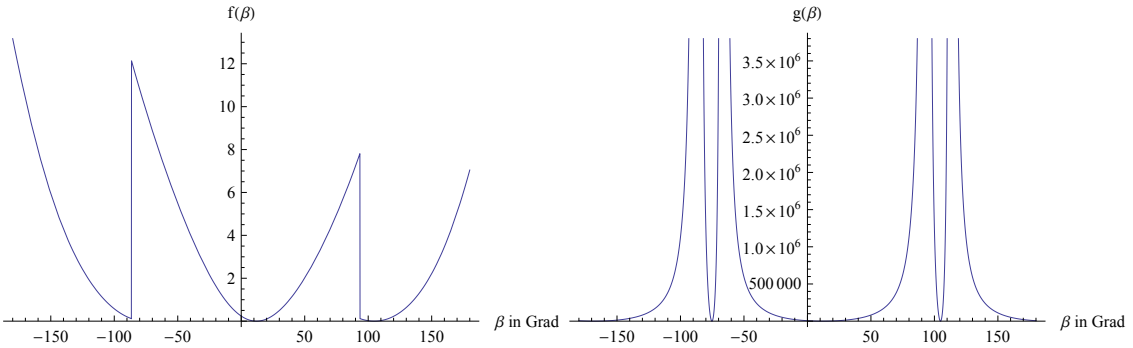


Abbildung 2.9.: *Funktionen f (links) und g (rechts) aus dem Abschnitt 2.2.3. Beide Funktionen wurden für den Spezialfall geplottet, dass der Abstand der Landmarke zum Roboter $d = 1000mm$ beträgt. Die Hohe der Landmarke ist $h_R = 350mm$, die Höhe der Kamera $h = 150mm$ und der Neigungswinkel der Kamera $\beta = 12°$.*

2.2.4. Korrektur des Rotationswinkels anhand des Horizonts

Bei den Verfahren aus den Abschnitten 2.2.1, 2.2.2 und 2.2.3 nehmen wir implizit an, dass die Kamera nicht um die optische Achse rotiert ist.

Die Rotation der Kamera wirkt sich dabei nicht nur auf die Bestimmung der Neigung aus, sondern auch auf die anschließende Bestimmung der Entfernung zu einzelnen Objekten, sofern diese sich nicht in der Bildmitte befinden.

Die Auswirkungen der Rotation auf die Bestimmung des Neigungswinkels werden ausführlich in dem Abschnitt 2.3.3 untersucht. Um die Rotation der Kamera um die eigene Achse zu berechnen, können wir die „Neigung“ der Objekte im Bild benutzen. Wenn wir als Beispiel eine Pylon-Landmarke der *4-Legged League*[3] betrachten, dann verläuft der Horizont stets orthogonal zu der Landmarke. Eine andere Möglichkeit die Neigung des Horizonts zu berechnen ist es die Höhe des Horizonts an mehreren Objekten abzutragen, z.B. an den beiden Pfosten eines Tores, wie in Abbildung 2.7 veranschaulicht. Aus der Neigung des Horizonts lässt sich nun die Rotation der Kamera um die eigene Achse sehr einfach berechnen, wenn wir den Horizont durch eine Gerade im Bild darstellen, dann entspricht die Rotation der Kamera genau dem Anstieg dieser Geraden.

2.2.5. Wissen über die kinematische Kette

In einigen Fällen ist die kinematische Kette des Roboters bekannt, sodass wir aus den Gelenkdaten die Position der Kamera bestimmen können. Das ist zum Beispiel beim Aibo Roboter der Fall. Hier lässt sich die gesamte Kameramatrix nur aus den Gelenkdaten bestimmen. Allerdings ist das Ergebnis zum Teil sehr ungenau. Das liegt zum einen daran, dass die Kontaktpunkte der Füße zum Boden nicht genau

[3]Liga im RoboCup in der nur Aibo-Roboter als Plattform eingesetzt werden dürfen

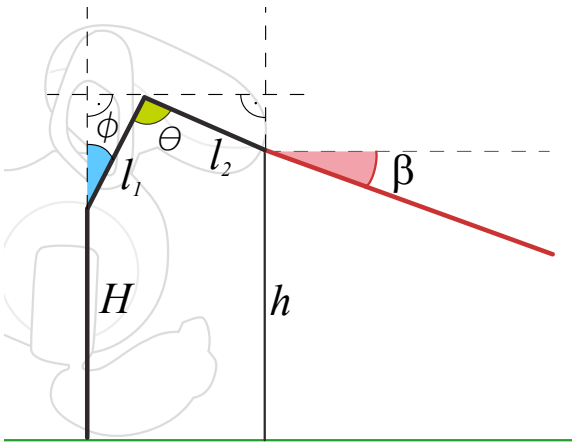


Abbildung 2.10.: *Zusammenhang der Höhe der Kamera mit dem Winkel des Nackengelenks eines Aibo ERS-7. Der Winkel des Nackengelenks ist mit ϕ bezeichnet.*

bestimmt werden können und zum anderen daran, dass einige Gelenksensoren sehr fehlerhafte Messwerte liefern. In diesem Abschnitt wollen wir darauf eingehen, wie man das Wissen der kinematischen Kette in Verbindung mit den oben vorgestellten Verfahren verwenden kann um insgesamt bessere Resultate zu erhalten.

Alle vorgestellten Verfahren liefern uns Zusammenhänge bzw. Abhängigkeiten zwischen unterschiedlichen Parametern der Kamera wie etwa dem Neigungswinkel und der Höhe, die sich aufgrund verschiedener Beobachtungen ergeben. Die kinematische Kette liefert uns ebenfalls Zusammenhänge dieser Parameter. Es ist also sehr naheliegend zu versuchen die interessanten Parameter so zu bestimmen, dass alle vorliegenden Zusammenhänge möglichst gut erfüllt (eingehalten) sind.

Meistens ist es so, dass mehr Parameter bestimmt werden müssen, als unabhängige Zusammenhänge vorliegen. Wir können jedoch alle Parameter der Kamera in Abhängigkeit von den Gelenkwinkeln der kinematischen Kette darstellen. Einige Gelenkwinkel können wir wiederum als Parameter auffassen und diese nun optimieren.

Beispielhaft haben wir im Fall des Aibo Roboters die Methode aus dem Abschnitt 2.2.1 (Verwendung von Referenzobjekten) dazu verwendet den Winkel des Nackengelenks zu korrigieren.

Anwendung auf den Aibo Roboter

Als eine der größten Quellen für die Fehler im Neigungswinkel der Kamera haben wir das Nackengelenk des Roboters (Aibo) identifiziert. Dieses Gelenk hat auch einen Einfluss auf die Berechnung der Höhe der Kamera. Bei dem in Abschnitt 2.2.1 beschriebenen Verfahren fließt die Höhe der Kamera jedoch in die Berechnungen ein, was dazu führt, dass die Fehler des Nackengelenks sich auf die Ergebnisse auswirken.

Wir wollen nun den Einfluss des Nackengelenks vollständig eliminieren. Dazu ist

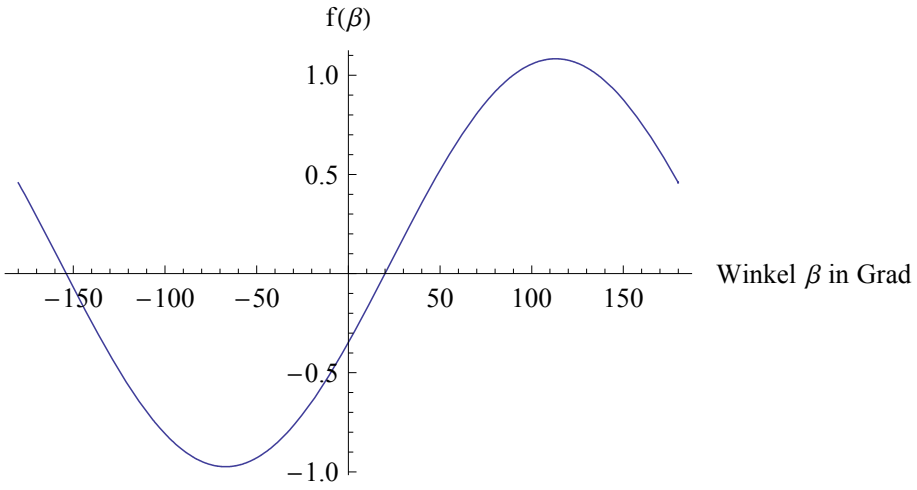


Abbildung 2.11.: *Beispiel der Funktion 2.40. Es wurden folgende Parameter verwendet: Höhe des Nackengelenks $H = 100mm$, Ballradius $\rho = 45mm$, Entfernung zum Ball $d = 1000mm$, Neigungswinkel $\beta = 20°$, $l_1 = 50mm$, $l_2 = 40mm$, $\theta = \frac{\pi}{2}$. Der Winkel β kann als Nullstelle der Funktion berechnet werden. Im zulässigen Bereich $[-90, 90]$ besitzt die Funktion f genau eine Nullstelle.*

jedoch das Wissen über den Zusammenhang zwischen dem Nackengelenk und der Höhe notwendig. Dieser Zusammenhang wird in der Abbildung 2.10 veranschaulicht.

Für die Höhe der Kamera lässt sich folgende Abhängigkeit vom Nackengelenk herleiten:

$$h(\phi) = H + l_1 \cdot \cos(\phi) - l_2 \cdot \sin(\frac{\pi}{2} - \theta - \phi) \tag{2.38}$$

Für den Neigungswinkel β ergibt sich

$$\beta = \pi - \theta + \phi. \tag{2.39}$$

Eingesetzt in die Formel aus 2.2.1 können wir mit dortigen Bezeichnungen folgende Funktion definieren

$$f(\beta) := \left(\frac{h(\theta + \beta - \pi) - \rho}{\rho} \cdot \sin(\alpha)\right) - \cos(\beta + \gamma - \frac{\pi}{2}) \tag{2.40}$$

Der Winkel ϕ kann jetzt als Nullstelle der Funktion f bestimmt werden. Auf diese Weise haben die Sensorwerte des Nackengelenks keinen Einfluss auf die Bestimmung der Entfernung von Objekten.

Korrektur der Rotation

Das Wissen der kinematischen Kette kann man natürlich auch dazu benutzen den Rotationswinkel der Kamera zu berechnen. Dadurch lassen sich die oben beschriebe-

nen Verfahren verbessern, indem die Höhe der Objekte nun bezüglich der richtigen Kamerakoordinaten berechnet werden kann. Was bei dieser Berechnung jedoch nicht berücksichtigt werden kann ist die Rotation des gesamten Körpers, von der die Kamera zweifellos betroffen ist. Eine solche Rotation entsteht zum Beispiel durch das „Wanken“ des Roboters von einer Seite zur anderen, das beim Laufen verstärkt auftritt. Die Rotation, die dabei entsteht, ist in der Regel nicht sehr groß. Laut Fehlerabschätzung im Abschnitt 2.3.3 liefern uns also die Verfahren aus vorherigen Abschnitten brauchbare Ergebnisse.

2.3. Fehleranalyse

Bei allen Methoden, die wir im vorherigem Abschnitt vorgestellt haben, wurden gewisse vereinfachende Annahmen getroffen. In den meisten Fällen wurde zum Beispiel die Rotation der Kamera um die eigene Achse vernachlässigt. In diesem Abschnitt wollen wir die Auswirkung der größten Fehlerquellen auf die oben vorgestellten Verfahren analysieren, um so Aussagen über die Güte der Ergebnisse machen zu können.

2.3.1. Winkelfehler bei der Horizont-Methode

Sehr oft ist die Höhe des Roboters nicht genau bekannt (z.B. wenn der Roboter läuft). Das Verfahren aus dem Abschnitt 2.2.3 ist besonders robust gegenüber solcher Fehler. Denn falls die Höhe des Roboters um h_e falsch eingeschätzt wird, so gilt für den Fehler β_e im resultierenden Rotationswinkel

$$\tan(\beta_e) = \frac{h_e}{d}, \tag{2.41}$$

wobei d die Entfernung zum Objekt ist, an dem der Horizont abgetragen wurde. Im Fall des Aibo Roboters würde sich also bei einem Fehler von $h_e = 3cm$ und einem Abstand von $d = 1m$ zum Tor ein Winkelfehler von gerade mal $\beta_e = 0.03$ ergeben (da arctan in der nähe der 0 nahezu linear verläuft). Insbesondere wird das Verfahren um so robuster, je weiter das Referenzobjekt entfernt ist.

2.3.2. Fehler bei der Strahlensatz-Approximation

In diesem Abschnitt untersuchen wir die Fehler, die bei dem im Abschnitt 2.2.3 beschriebenen Verfahren entstehen. In diesem Abschnitt wird folgende Formel für die Berechnung des Neigungswinkels β vorgeschlagen

$$\beta = \arctan\left(\frac{h-H}{H}\tan(\delta) - \frac{h}{H}\tan(\gamma)\right) \tag{2.42}$$

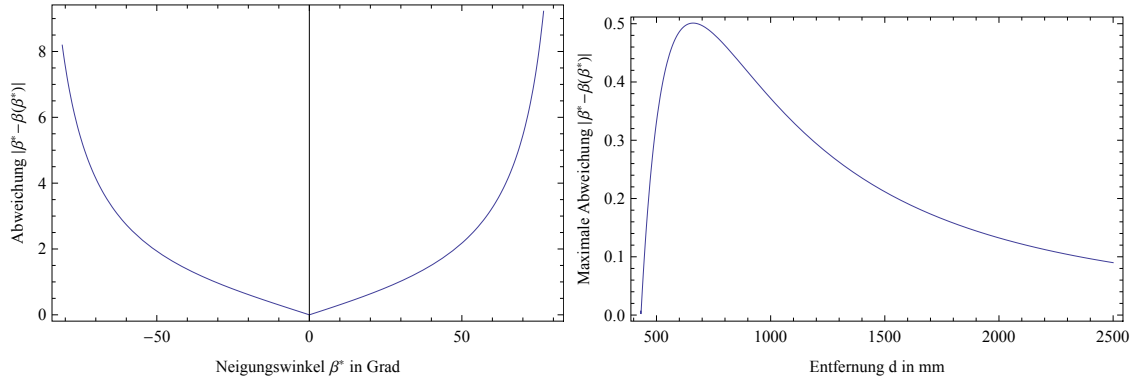


Abbildung 2.12.: *(links) die Abweichung $|\beta^* - \beta(\beta^*)|$ des echten Neigungswinkels β^* von dem Winkel berechnet nach der Methode aus dem Abschnitt 2.2.3. (rechts) die maximale mögliche Abweichung $|\beta^* - \beta(\beta^*)|$ in Abhängigkeit von der Entfernung zu der Landmarke. Beide Kurven sind berechnet für $H = 350mm$ und $h = 150mm$.*

wobei h die Höhe der Kamera ist, H die Höhe der Landmarke, δ bzw. γ der Winkel vom Zentrum des Bildes zur Unterkante bzw. Oberkante der Landmarke im Bild. Die Bezeichnungen werden in der Abbildung 2.8 veranschaulicht.

Ist der echte Neigungswinkel β^* gegeben, so können wir die Winkel γ und δ wie folgt schreiben

$$\gamma = \tan\left(\frac{H-h}{d}\right) + \beta^* \tag{2.43}$$

und

$$\delta = \tan\left(\frac{h}{d}\right) - \beta^* \tag{2.44}$$

wobei d die Entfernung zu der Landmarke ist. Somit können wir den berechneten Winkel β als Funktion des echten Winkels auffassen $\beta = \beta(\beta^*)$. Die Abbildung 2.12(links) veranschaulicht die Abweichung

$$|\beta^* - \beta(\beta^*)| \tag{2.45}$$

für den Fall $d = 1000mm$, $h = 150mm$ und $H = 350mm$, ähnlich einer Situation in der Aibo-Liga. Bei der Betrachtung der Abweichung ist jedoch zu beachten, dass der Winkel β^* nicht in seinem maximalen Rahmen variieren kann, ohne dass die Landmarke zumindest teilweise das Bild verlässt. Genauer gesagt, es muss gelten

$$-\frac{\theta - \gamma + \delta}{2} \leq \beta^* \leq \frac{\theta - \gamma + \delta}{2} \tag{2.46}$$

wobei θ der vertikale Öffnungswinkel der Kamera ist. Diese Bedingung ist ziemlich offensichtlich, denn $\gamma + \delta$ beschreibt den Winkel der gesamten Landmarke im Bild.

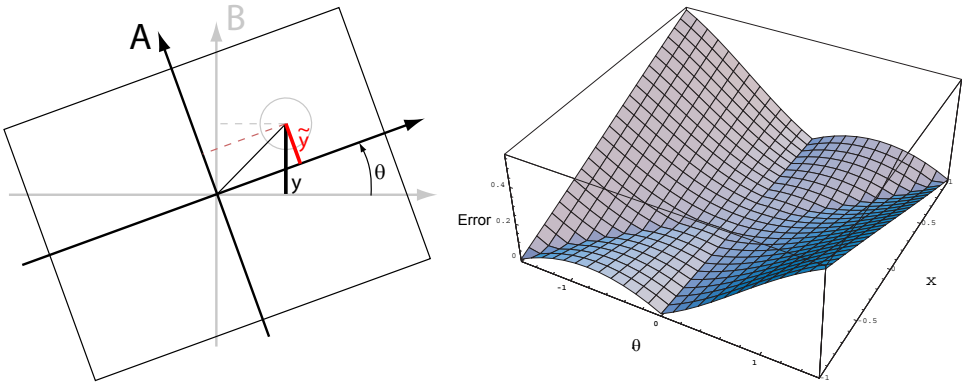


Abbildung 2.13.: *(links) Kamera rotiert um die optische Achse im Bezug zu einem nicht rotierten Koordinatensystem. Das Koordinatensystem (A) stellt das tatsächliche Koordinatensystem der Kamera dar, (B) ist das nicht rotierte Koordinatensystem. Die Koordinate y ist notwendig für die Bestimmung der Entfernung, die aus dem Bild bestimmte (und für die Berechnungen benutzte) Koordinate $\tilde{y}$ unterscheidet sich jedoch von y, falls $\theta \neq 0$. (rechts) Abweichung $|\tilde{\beta} - \beta|$ die durch die Vernachlässigung der Rotation der Kamera um die eigene Achse um den Winkel θ entsteht. Es wurde für die Position $y = 1mm$ (nahezu die maximal mögliche y-Position auf dem CCD-Chip der Kamera von Aibo ERS7), und für die Fokallänge $f = 3.5mm$ angenommen, wobei θ und x variiert wurden.*

Aus dem Verlauf der Abweichung lässt sich sofort schließen, dass die maximale Abweichung stets für $\beta^* = \pm\frac{\theta-\gamma+\delta}{2}$ angenommen wird. Wir können den Winkel der gesamten Landmarke im Bild wie folgt in Abhängigkeit von der Entfernung darstellen

$$\gamma + \delta = \arctan\left(\frac{h}{d}\right) + \arctan\left(\frac{H-h}{d}\right) \tag{2.47}$$

Diese Formel lässt sich sofort aus der Abbildung 2.12 ableiten. Die Abbildung 2.12 (rechts) veranschaulicht die maximale Abweichung in Grad für die Situation des Aibo Roboters (der Öffnungswinkel beträgt in diesem Fall $\theta = 44°$). An dem Verlauf der Funktion lässt sich sofort erkennen, dass die maximale überhaupt mögliche Abweichung 0.5 Grad nicht überschreitet. Ab einer bestimmten Entfernung sinkt der Fehler sogar.

Insgesamt können wir also sagen, dass die Fehler, die durch die vereinfachte Anwendung des Strahlensatzes entstehen, vernachlässigt werden können.

2.3.3. Untersuchung des Rotationsfehlers

Der Neigungswinkel der Kamera ist essentiell für die Berechnung der Entfernung zu anderen Objekten. Daher konzentrieren sich alle oben beschriebenen Verfahren auf die Korrektur des Neigungswinkels.

In der Tat hat aber unter Umständen der Rotationswinkel der Kamera ebenfalls einen großen Einfluss auf das Ergebnis. In den Verfahren aus den Abschnitten 2.2.1, 2.2.2 und 2.2.3 wird der Rotationswinkel nicht beachtet, hier wollen wir nun untersuchen, welche Auswirkungen das auf das Ergebnis dieser Verfahren hat (wie stark wirkt sich das auf die Korrektur aus).

Wir betrachten alle Koordinaten bezüglich des Bildzentrums. Sei

$$p = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.48}$$

das Zentrum des Balls im nicht rotierten Bild, und θ die Rotation der Kamera um die eigene Achse. Für die Korrektur der Neigung brauchen wir die y-Position des Ballzentrums im Bild. Wir rotieren den Punkt p um den Winkel $-\theta$ und erhalten die Position des Ballzentrums, wie wir sie in einem realen Bild ablesen können:

$$\tilde{p} := \begin{pmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \end{pmatrix} \tag{2.49}$$

D.h. die gemessene Höhe des Ballzentrums im Bild ist

$$\tilde{y} = x \cdot \sin\theta + y \cdot \cos\theta. \tag{2.50}$$

Es ist also offensichtlich, dass die Stärke des Einflusses der Rotation von der Entfernung des Balls zum Zentrum des Bildes abhängig ist (was man sich auch sehr anschaulich vorstellen kann).

Mit den Bezeichnungen aus dem Abschnitt 2.2.1 können wir nun schreiben

$$\gamma = \arctan\frac{\tilde{y}}{f} \tag{2.51}$$

wobei f die Brennweite der Kamera ist. Für das Gesamtergebnis folgt dann

$$\beta = \arccos\left(\frac{h-\rho}{\rho} \cdot \sin(\alpha)\right) + \arctan\frac{\tilde{y}}{f}. \tag{2.52}$$

Die Abbildung 2.13(rechts) veranschaulicht den resultierenden Fehler im Fall $\theta \neq 0$. Wie man leicht sieht, kann der Fehler vernachlässigt werden, wenn der Winkel θ in der Nähe der Null liegt. Man kann also schließen, dass die Methoden trotz Vernachlässigung der Rotation der Kamera um die eigene Achse annehmbare Resultate liefern, wenn der Rotationswinkel klein genug ist.

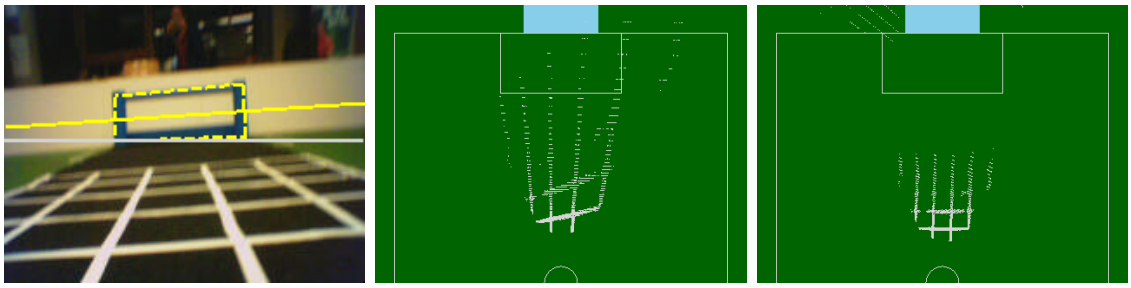


Abbildung 2.14.: *(links) die Szene aus der Sicht des Roboters, (mitte) Projektion des Gitters mit Hilfe der Kameramatrix aus den Gelenkdaten, (rechts) Projektion des Gitters mit Hilfe der korrigierten Matrix.*

2.4. Experimente

Um die vorgestellten Ideen zu evaluieren, haben wir eine Reihe von Versuchen auf unterschiedlichen Plattformen durchgeführt. Dieser Abschnitt unterteilt sich in zwei Teile. Im ersten Teil untersuchen wir die Ergebnisse der Kalibrierungsmethoden indem wir die Fehler in der Kameramatrix mit Hilfe von Verzerrungen in Projektionen visualisieren. Im zweiten Abschnitt führen wir einige Experimente durch, bei denen eine konstante Entfernung zu einem Referenz-Objekt als Maß für die Korrektheit der Ergebnisse verwendet wird.

2.4.1. Projektions-Experimente

In diesem Abschnitt stellen wir Experimente vor, bei denen die Korrektheit der Kameramatrix, anhand der Genauigkeit der mit dieser Matrix erzeugten Projektionen, beurteilt wird. Dabei wird ein Muster auf dem Boden von der Kamera aufgenommen und mit Hilfe der Kameramatrix auf die imaginäre Boden-Ebene projiziert. Ist die Kameramatrix ungenau, so treten in der Projektion Verzerrungen auf. Auf diese Weise kann die Güte der Matrix beurteilt werden.

Horizont-Basierte Korrektur des Neigungs- und Rotationswinkels

In unserem ersten Experiment befindet sich der Roboter im Zentrum des Feldes und ist in Richtung des blauen Tores ausgerichtet. Vor dem Roboter ist ein quadratisches weißes Gitter platziert. Die Zellen des Gitters haben die Größe von $20cm \times 20cm$. Während des Experiments läuft der Roboter auf der Stelle und schaut vorwärts in Richtung des blauen Tores. Dabei wird die Kameramatrix aus den Gelenkdaten berechnet und eine Korrektur durchgeführt, indem die Neigungs- und Rotationswinkel mit Hilfe des Tores im Bild berechnet werden (nach dem Verfahren aus dem Abschnitt 2.2.3). Die Abbildung 2.14 (links) zeigt die Situation aus der Sicht des Roboters. Das von dem Roboter aufgenommene Bild wird jeweils mit Hilfe der beiden Matrizen auf die imaginäre Feld-Ebene projiziert.

Wenn die Werte der Kameramatrix nicht mit der Realität übereinstimmen, treten Verzerrungen auf, d.h. das Seitenverhältnis der Zellen stimmt nicht mehr überein und die Ecken sind nicht mehr rechtwinklig. Alle diese Effekte lassen sich in dem Fall der aus den Gelenkdaten berechneten Kameramatrix verstärkt beobachten (Abbildung 2.14 (Mitte)). Im Fall der korrigierten Matrix treten dagegen weitestgehend keine Verzerrungen auf.

Bestimmung des Neigungswinkels anhand der Ecken von Feldlinien

In diesem Experiment werden die Ecken der Feldlinien dazu verwendet den Neigungswinkel der Kamera zu bestimmen wie in dem Abschnitt 2.2.2 beschrieben.

Das Experiment besteht aus zwei Teilen. Zunächst steht der Roboter auf der Stelle und schaut auf eine Feld-Ecke von innen. Während des Experiments werden die hinteren Beine des Roboters um ca. 10cm angehoben, was in einem Neigungswinkel von ca. 30° resultiert. Diese Winkeländerung spiegelt sich nicht in den Gelenken des Roboters wider und wird also nicht in der Kameramatrix berücksichtigt, wenn sie nur anhand der kinematischen Kette berechnet wird. Diese Situation entspricht im gewissen Sinne einem *kidnapped robot problem*, da der Roboter hier unfreiwillig bewegt wird. In einem RoboCup Spiel passiert es oft, dass ein Roboter durch Kollision mit anderen verschoben oder angehoben wird. Die Abbildung 2.15 Ia) zeigt die Scene aus der Sicht des Roboters.

Im zweiten Experiment läuft der Roboter auf der Stelle und schaut auf eine Feldecke von außen. Dadurch werden die Fehler simuliert, die beim normalen Laufen entstehen. Die Abbildung 2.15 IIa) zeigt die Situation aus der Sicht des Roboters.

In beiden Experimenten ändert sich die Entfernung zu der Ecke nicht. Um die Ergebnisse der Experimente zu visualisieren, wurden die Bilder der Ecken auf das Feld projiziert. Für die Projektion wurde jeweils die aus den Gelenkdaten berechnete Kameramatrix verwendet und die Kameramatrix korrigiert anhand der gesehenen Ecke mit dem Verfahren aus dem Abschnitt 2.2.2. Die Abbildung 2.15 veranschaulicht die Ergebnisse. In den beiden Teilexperimenten ist zu beobachten, dass bei der Projektion mit der nicht korrigierten Kameramatrix Verzerrungen auftreten, insbesondere wird der Abstand zu der Ecke falsch widergespiegelt. Bei der Berechnung der Kameramatrix aus den Gelenkdaten wird davon ausgegangen, dass die Füße Kontakt mit dem Boden haben. Die im ersten Experiment manuell erzeugte Neigung kann also nicht bestimmt werden und fließt als Fehler ein. Das resultiert in einer sehr starken Verzerrung der Ecke.

Das zweite Experiment ist näher zu der Realität. Die Fehler im Neigungswinkel der auf den Gelenkdaten berechneten Kameramatrix entstehen hier infolge der Laufbewegung des Roboters (Schwanken des Nackengelenks, unbekannte Kontaktpunkte zum Boden, etc.).

Bei der Projektion der Ecke mit der korrigierten Kameramatrix lassen sich jedoch

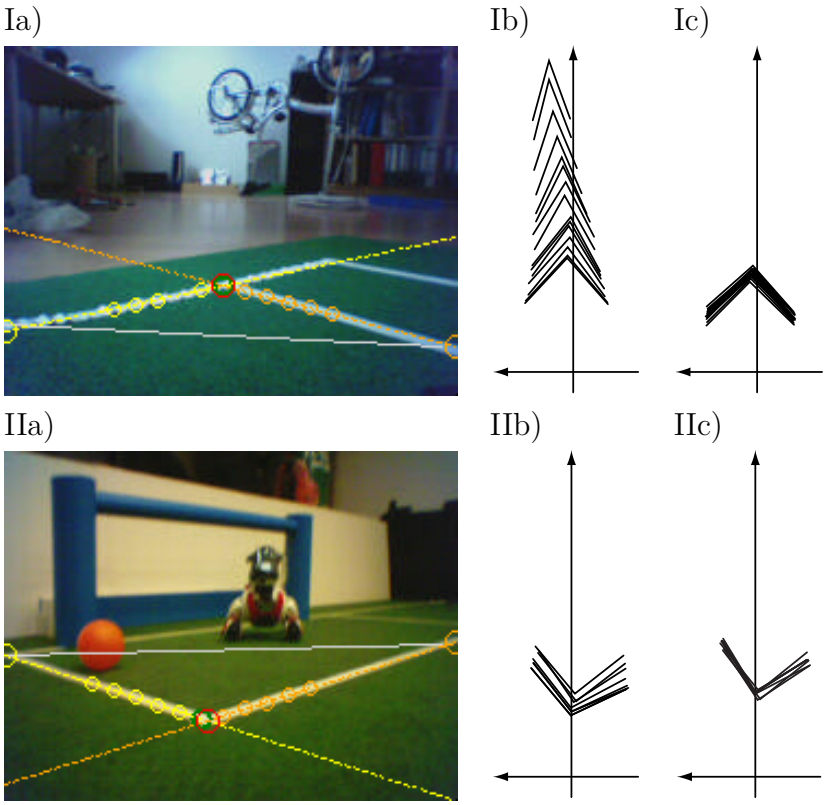


Abbildung 2.15.: *Diese Abbildung illustriert die Korrektur des Neigungswinkels der Kamera mit Hilfe der Ecken der Feldlinien. Die Bilder Ia) und IIa) zeigen die Situation während des Experiments aus der Sicht des Roboters. Im ersten Experiment wurden die hinteren Beine des Roboters manuell angehoben, sodass die resultierende Neigung nicht aus den Gelenkdaten berechnet werden konnte. Die Abbildungen Ib) und Ic) zeigen die Projektion der Ecke im Bild auf den Boden, jeweils mit der Kameramatrix die nur aus den Gelenkdaten berechnet wurde und mit der korrigierten Kameramatrix. IIb) und IIc) zeigen die nicht-korrigierte und korrigierte Projektionen der gesehenen Ecke, wobei der Roboter auf der Stelle lief.*

Abbildung 2.16.: *Die Situation des Experiments aus der Sicht des Roboters. Das Tor und der Ball werden verwendet um die Entfernung zum anderen Roboter im Bild zu bestimmen.*

in beiden Experimenten weitestgehend keine Verzerrungen beobachten. Insbesondere wird die Entfernung zu der Ecke richtig bestimmt.

2.4.2. Distanz-Experiment

In diesem Abschnitt stellen wir Experimente vor in denen wir die Methoden testen, die Referenzobjekte benutzen um die Entfernung zu berechnen. Es wurden insbesondere Experimente auf einem Aibo und einem humanoiden Roboter durchgeführt.

Experiment auf dem Aibo Roboter

In diesem Experiment bestimmen wir peilungsbasiert die Entfernung zu einem anderen Roboter, wobei die Parameter der Kamera mit verschiedenen Verfahren bestimmt wurden. Das ermöglicht uns die Verfahren untereinander zu vergleichen. Die Situation des Experiments stimmt mit der Situation des ersten Experiments überein. Jetzt befindet sich allerdings kein Kalibrierungsteppich vor dem Roboter. Außerdem ist ein weiterer Roboter und ein Ball im Blickfeld.

In diesem Experiment korrigieren wir die Kameramatrix jeweils mit Hilfe des Balls (direkt und indirekt) und des Tores. Als Test bestimmen wir dann mit den korrigierten Matrizen peilungsbasiert die Entfernung zum anderen Roboter. Da der Roboter auf der Stelle läuft und der andere Roboter sich nicht bewegt, bleibt die tatsächliche Entfernung stets konstant. Die berechnete Entfernung variiert jedoch aufgrund von Fehlern. In der Abbildung 2.17 sind mit verschiedenen Verfahren berechnete Entfernungen über einer Zeit von ca. 30s abgetragen.

Besonders hervorstechend sind die Abweichungen im nicht korrigierten Fall. Das beste Ergebnis liefert dagegen das Horizont-basierte Verfahren. Die beiden Verfah-

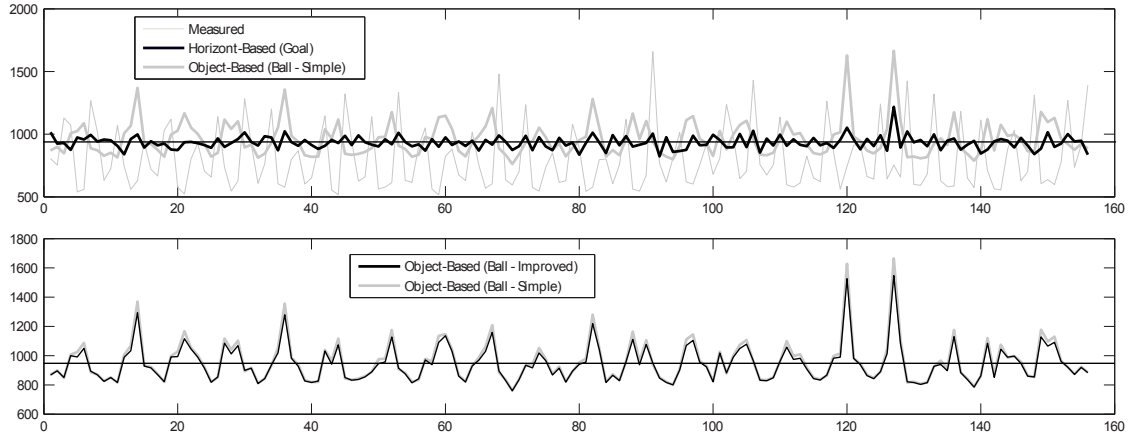


Abbildung 2.17.: *(oben) Entfernung berechnet mit der aus den Gelenkdaten bestimmten Kameramatrix im Vergleich zu der Entfernung die bestimmt wurde jeweils unter Benutzung des Balls als Referenzobjekt (beschrieben im Abschnitt 2.2.1) und mit Hilfe des Horizonts, wie im Abschnitt 2.2.3. (unten) Vergleich der Ergebnisse der Methode, bei der ein Ball als Referenzobjekt benutzt wird und der Kombination dieser Methode mit dem Wissen der kinematischen Kette wie im Abschnitt 2.2.5 dargestellt.*

ren, die den Ball als Referenzobjekt benutzen, liefern nahezu identische Ergebnisse, in einem akzeptablen Fehlerbereich.

Experiment auf einem Humanoiden Roboter

Alle bisherigen Experimente wurden auf einem Aibo durchgeführt. Für dieses Experiment wurde ein humanoider Roboter auf der Bioloid-Basis benutzt. Diese Roboter wurden von dem Humanoid Team Humboldt für die Teilnahme am RoboCup eingesetzt [15].

Im Unterschied zum Aibo besitzt der humanoide Roboter mehr Gelenke in der kinematischen Kette zwischen Boden und Kamera und ist weniger stabil, da er nur zwei Kontakt-Punkte zum Boden besitzt. Des Weiteren macht ein höher platzierter Schwerpunkt den Roboter zusätzlich instabil.

Aufgrund der Instabilität des Roboters variiert die Neigung des Körpers während des Laufens sehr stark. Zusammen mit den Fehlern der Gelenke-Sensoren führt es dazu, dass der aus der kinematischen Kette berechnete Neigungswinkel der Kamera sehr ungenau ist.

Die Abbildung 2.18 veranschaulicht die berechnete Entfernung zum Ball über die Zeit, während der Roboter sich dem Ball nähert. Die Entfernung wurde jeweils basierend auf der Größe und der Peilung berechnet. Wie man im Graph leicht sehen kann, oszillieren die Ergebnisse der auf Peilung basierten Berechnung sehr stark. Die

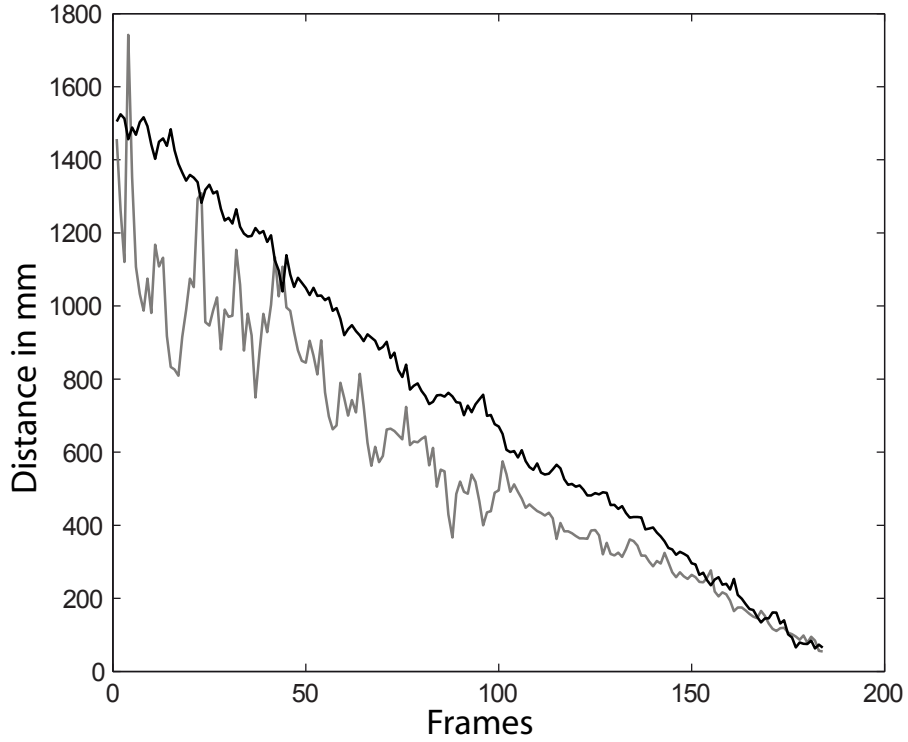


Abbildung 2.18.: *Die berechnete Entfernung zum Ball, während der Roboter sich dem Ball nähert. (schwarz) Entfernung basiert auf der Größe des Balls; (grau) Entfernung basiert auf der Peilung, wobei der aus den Gelenkdaten berechnete Neigungswinkel der Kamera verwendet wurde;*

Oszillationen nehmen mit größerer Entfernung zu (man kann es als experimentellen Nachweis der Ergebnisse aus dem Abschnitt betrachten). Insbesondere lässt sich erkennen, dass die Benutzung des Balls als Referenzobjekt bessere Resultate für die Berechnung der Entfernung zu einem Objekt verspricht, als die peilungsbasierte Methode, bei der die aus den Gelenkdaten berechnete Neigung der Kamera benutzt wird.

Im Rahmen des nächsten Experiments berechnen wir die Entfernung zu einem fixierten Linienpunkt während der Roboter auf der Stelle läuft. D.h. die tatsächliche Distanz zum Punkt ändert sich nicht. Dabei wollen wir untersuchen, wie sich die Schwankungen des Roboters, die beim Laufen entstehen, auf die Berechnung der Entfernung auswirken. Dabei wird die Entfernung basierend auf der Peilung bestimmt, wobei zum Vergleich jeweils der aus den Gelenkdaten und mit Hilfe des Balls als Referenzobjekt berechnete Neigungswinkel der Kamera benutzt wird.

Die Ergebnisse werden in der Abbildung 2.19 veranschaulicht. Während die ohne Korrektur berechnete Entfernung stark schwankt, bleibt das Resultat des korrigierten Verfahrens weitestgehend stabil.

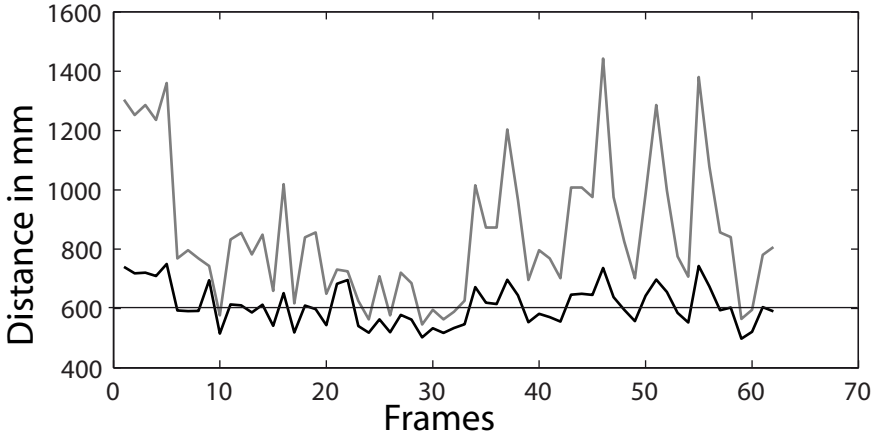


Abbildung 2.19.: *Entfernung zu einem Linienpunkt berechnet mit zwei unterschiedlichen Methoden. (schwarz) Entfernung berechnet mit Hilfe des Balls als Referenzobjekt; (grau) Entfernung basiert auf der Peilung, wobei der aus den Gelenkdaten berechnete Neigungswinkel der Kamera verwendet wurde;*

2.5. Zusammenfassung und Auswertung

Viele spezielle Eigenschaften von Objekten können dazu verwendet werden, die räumlich Relationen zwischen dem Roboter und diesen Objekten zu bestimmen. Diese Eigenschaften können auch allgemeiner als Referenzen dazu verwendet werden um auch Relationen zu anderen Objekten zu bestimmen.

Wir haben exemplarisch eine Reihe konkreter Möglichkeiten untersucht die externen Parameter der Kamera mit Hilfe von Referenz-Objekten in der Umgebung des Roboters zu bestimmen bzw. zu korrigieren. So kann die Positionsbestimmung anderer Objekte verbessert wird. Besonderes Interesse gilt dabei dem Neigungswinkel der Kamera, der essentiell für die peilungsbasierte Entfernungsbestimmung ist, aber auch gleichzeitig am stärksten von Fehlern betroffen zu sein scheint. Konkret haben wir die Größe und Form von Objekten (am Beispiel eines Balls und der Feldlinienecken), dazu verwendet den Neigungswinkel zu bestimmen. Als weitere Möglichkeit haben wir vorgestellt, wie der Horizont an hohen Objekten bestimmt werden kann. Mit Hilfe des Horizonts lassen sich dann der Neigungswinkel und die Rotation an der optischen Achse der Kamera leicht bestimmen.

Es wurden Fehlerbetrachtungen durchgeführt um die Robustheit der Ansätze zu überprüfen. Insbesondere wurden dabei die Auswirkungen der Aspekte untersucht, die als vereinfachende Annahmen in die Algorithmen eingehen, wie etwa die Vernachlässigung der Rotation der Kamera um die optische Achse.

Alle Algorithmen wurden implementiert und auf mehreren Roboter-Systemen getestet. In den Experimenten haben haben wir gesehen, dass durch solche gezielte

Nutzung objektspezifischer Eigenschaften die Wahrnehmungsfähigkeit des Roboters verbessert werden kann.

Insgesamt lässt sich eine allgemeine Vorgehensweise herausstellen: es wird ein Zusammenhang zwischen einigen Größen in Form einer Funktion definiert, in der gemessene und unbekannte Größen als Parameter vorkommen. Wenn genügend Größen bekannt sind, dann können andere analytisch bestimmt werden. Bei den vorgestellten Verfahren wurde immer nur eine Größe (meistens der Neigungwinkel) als Unbekannte bestimmt. Es ist aber offensichtlich denkbar dass mehrere Parameter simultan als Lösung (oder Optimum) eines Gleichungssystems bestimmt werden. Dabei könnte z.B. gleichzeitig die Position des Roboters, die Relationen zu Objekten oder auch die Gelenkwerte als Parameter auftauchen. Das resultierende Optimierungsproblem kann auch als Constraint Erfüllungs-Problem formuliert werden, wie wir in den nachfolgenden Kapiteln sehen werden.

3. Grundlagen der Constraint-Modellierung

Die Informationen die dem Roboter zur Verfügung stehen, enthalten viele Redundanzen. Insbesondere visuelle Daten enthalten viele nützliche Informationen wie die Größe und die Erscheinung beobachteter Objekte, Peilungswinkel, Entfernungen und andere Beziehungen zwischen Objekten, usw.. Die klassischen Algorithmen nutzen normalerweise nur einen kleinen Teil dieser Information. Die Ursache dafür könnte darin liegen, dass die meisten Algorithmen ursprünglich für Abstandsmessungen konzipiert wurden.

Aktuelle Verfahren verwenden meist stochastische Ansätze, die auf der Bayesschen Theorie beruhen (vgl. [13]). Sehr verbreitete Beispiele dafür sind Kalman-Filter [24, 28] und Partikelfilter [2, 14, 3, 32]. Die Vorteile dieser Verfahren liegen in der einfachen Implementierbarkeit und Robustheit. Diese Verfahren haben aber auch Einschränkungen. Die Leistungsfähigkeit solcher Filter hängt stark davon ab wie gut die zugrunde liegenden stochastischen Modelle die Wirklichkeit widerspiegeln. So kann der Kalman-Filter nur Gauss-Verteilungen repräsentieren. Partikelfilter haben diese Einschränkung nicht, benötigen aber eine große Anzahl von Partikeln um komplizierte Verteilungen zu repräsentieren. Unter bestimmten Umständen können diese Einschränkungen zu großen Nachteilen führen. Ein weiteres Problem stellt die große Berechnungskomplexität der Bayesschen Methoden (Gitter, Partikel) dar. Die Kalman-Filter haben das Problem zwar nicht, basieren aber auf einer festen Annahme über die zugrunde liegende stochastische Verteilung.

Ähnlich dem Kalman-Filter besitzen Constraints die Fähigkeit simultan die Positionen verschiedener Objekte und Beziehungen zwischen ihnen zu modellieren. Andererseits können damit aber auch sehr komplizierte Modelle beschreiben werden. Weitere Aspekte, die die Constraints zu einem vielversprechenden Ansatz machen sind:

- einfache Darstellung mehrdimensionaler Daten
- einfach zu kommunizieren
- können große Bereiche beschreiben
- formale Definition

Allerdings haben die auf den Constraints basierende Methoden ein besonderes Problem mit Inkonsistenz. Wenn die Messungen verrauscht sind, kann es passieren,

dass es keine Belegung gibt, die mit allen Constraints konsistent ist. Nach unserer Definition wäre in diesem Fall der Schnitt aller Constraints leer. Normalerweise bedeutet die Inkonsistenz eines CSP, dass keine Lösung existiert. Aber in unserem Fall besitzt der Roboter und andere Objekte eine reale Position in der Welt. Die Inkonsistenz wird nur durch Fehler in den Sensordaten verursacht. Um die Qualität der Constraint-Netze zu analysieren, wurden entsprechende Maße eingeführt (in [5, 10]). Wir diskutieren sie nochmal genauer im Abschnitt 3.2.

Aus der Sicht eines Navigationproblems könnte es sein, dass nur einige der Variablen von Interesse sind. Das ist zum Beispiel dann der Fall, wenn nur die Orientierung des Roboters von Interesse ist, aber nicht seine genaue Position. Nichtsdestotrotz müsste in diesem Fall das gesamte Problem gelöst werden um die Lösung zu finden.

Im Fall der Roboternavigation existiert in der Realität immer eine Lösung des Problems, nämlich die realen Positionen der Objekte. Das hat einen Einfluss auf die Interpretation der Lösungen und der Inkonsistenzen des Constraintsystems (vgl. Abschnitt 3.3).

Der Einsatz von Constraint-basierter Modellierung für Lokalisierungsaufgaben wurde bereits in anderen Arbeiten untersucht [34]. In [25, 33] wird eine auf Intervall-Analysis [17] basierende Lokalisierung für ein Fahrzeug vorgestellt. Als Sensordaten dienen hier Ultraschallsensoren. Eine SLAM Karten-Bildung mit Constraint-basierter Modellierung wird in [12, 27] vorgestellt. In [1, 11] werden grundlegende Techniken für Constraintpropagierung diskutiert, die wir in späteren Kapiteln aufgreifen. Weitere Verfahren wie etwa in [16, 21, 19] fassen die Lokalisierung als ein Minimierungsproblem auf, was auf natürliche Weise im Zusammenhang mit der Constraint-basierter Lokalisierung steht.

Teile der folgenden Abschnitte entstanden in einer Zusammenarbeit mit Daniel Göhring. Die Idee wurde von Prof. Dr. Hans-Dieter Burkhard vorgeschlagen. Von ihm stammen auch die wesentlichen Teile der formalen Grundlagen und die Analyse der Qualität der Constraint-Netze.

Die Resultate wurden in einer Reihe von Publikationen veröffentlicht. Die erste Idee und Formulierung der Maße für die Qualität der Constraint-Netze wurde bereits 2007 in [5] veröffentlicht. In [4] wird Kommunikation von Constraints und darauf basierende kooperative Lokalisierung beschrieben. In [8] wird der erste Propagierungs-Algorithmus vorgeschlagen. Die ersten Experimente im Simulator wurden in [10, 7] präsentiert. Das erste Experiment auf einem Aibo Roboter wird in [6] vorgestellt. Die letzte Arbeit [9] beschreibt Experimente auf einem Nao Roboter und schlägt einen verbesserten Algorithmus für die Propagierung von Constraints vor.

Als erstes werden die Constraints formal eingeführt. Im zweiten Abschnitt diskutieren wir dann einige Optimalitätskriterien für Constraint-Netze, und definieren die Begriffe der Inkonsistenz- und Mehrdeutigkeits-Maße. Im nachfolgenden Abschnitt stellen wir die Techniken für die Propagierung von Constraints vor und diskutieren wichtige Eigenschaften der Propagierungsfunktionen. Im vierten Abschnitt diskutieren wir speziell die Intervall-Constraints und einige Techniken für die Optimierung

der algorithmischen Umsetzung der Constraint Propagierung. Am Ende wird diese Kapitel mit einer Zusammenfassung abgeschlossen.

3.1. Formale Definition

Sei $\mathcal{V} := \{v_1, ..., v_n\}$ eine Menge von *Variablen*. Zu jeder Variable $v \in \mathcal{V}$ bezeichne $Dom(v)$ die Menge aller möglichen Werte von v. Basierend darauf definieren wir das *Universum*

$$\mathcal{U} := Dom(v_1) \times \cdots \times Dom(v_n). \tag{3.1}$$

Eine *Belegung* $\beta \in \mathcal{U}$ weist jeder Variable einen Wert zu.

Wir definieren die Constraints über alle Variablen $v_1, ..., v_n$, insbesondere auch dann wenn einige der Variablen nicht von dem Constraint gebunden sind. Das vereinfacht mengentheoretische Betrachtungen von Constraints. Des Weiteren beschränken wir uns bei den Wertebereichen von Variablen $Dom(v)$ auf reellwertige Intervalle, d.h.

$$\forall v \in \mathcal{V} : Dom(v) \subseteq \mathbb{R}. \tag{3.2}$$

Insbesondere gilt dann auch $\mathcal{U} \subseteq \mathbb{R}^n$

Definition 3.1.1 *(Constraints)*

1. *Ein* **Constraint** C *über den Variablen* $\mathcal{V}$ *ist eine Teilmenge* $C \subseteq \mathcal{U}$.
2. *Eine Zuweisung/Belegung* β *von Werten zu den Variablen* $\mathcal{V}$*, d.h.* $\beta \in \mathcal{U}$*, heißt eine* **Lösung** *von* C *genau dann, wenn* $\beta \in C$.

Definition 3.1.2 *(Constraint-Netz)*

1. *Ein* **Constraint-Netz** $\mathcal{C} = \{C_1, ..., C_n\}$ *über den Variablen* $\mathcal{V}$ *ist gegeben durch eine endliche Menge von Constraints über diesen Variablen.*
2. *Eine Zuweisung* $\beta \in \mathcal{U}$ *ist eine* **Lösung** *von* $\mathcal{C}$ *genau dann, wenn* β *eine Lösung für alle Constraints* $C \in \mathcal{C}$ *ist, d.h. wenn* $\beta \in \bigcap \mathcal{C}$.
3. *Ein Constraint-Netz* $\mathcal{C}$ *heißt* **inkonsistent***, wenn es keine Lösung besitzt, d.h. wenn* $\bigcap \mathcal{C} = \emptyset$.

Als Constraint-Erfüllungs-Problem (CSP[1]) bezeichnen wir das Problem des Finden einer Lösung in einem gegebenen Constraint-Netz $\mathcal{C}$. Eine Lösung des Problems ist nach unserer Definition ein Punkt im Universum $\mathcal{U}$, d.h. eine Belegung aller Variablen mit Werten.

Constraints sind Modelle der Relationen (Einschränkungen) zwischen Objekten in der Szene. Die Information kann von einem Sensor, Kommunikation mit anderen Robotern oder aus dem Wissen über die Welt gewonnen werden. Im Allgemeinen kann

[1] Constraint Satisfaction Problem

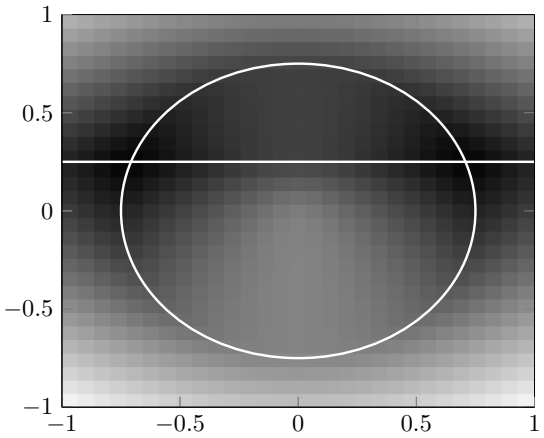


Abbildung 3.1.: *Punkte mit der gleichen Helligkeit haben die gleich Distanz s zum Constraint-Netz. Je dunkler die Punkte umso kleiner ist s. Die Mengen $P_{\mathcal{C}}(s)$ bestehen aus Punkten deren Helligkeit dem Wert s oder kleiner entsprechen (dunklere Punkte).*

ein Constraint einen Teilraum beliebiger Dimension beschreiben, z.B. den gesamten Strafraum, alle möglichen Positionen eines verdeckten Objektes, etc.. Constraint müssen nicht notwendig zusammenhängend sein, im Fall ununterscheidbarer Landmarken können die Constraints aus mehreren disjunkten Teilmengen bestehen (z.B. im Fall von Linien entstehen mehrere disjunkte Boxen). Weitere Constraints können aus den Geschwindigkeiten gewonnen werden, denn die Änderungen der Position sind durch die Richtung und die Geschwindigkeit der Bewegung beschränkt.

3.2. Optimalitätskriterien für Constraint-Netze

Nur wenn alle Messungen fehlerfrei sind, dann können wir erwarten eine wohldefinierte Lösung des Navigationsproblems zu erhalten, d.h. genaue Positionen interessanter Objekte. Zusätzlich müssen ausreichend Constraints vorhanden sein, um die Lösungsmenge genügend einzuschränken, anderenfalls erhalten wir eine mehrdeutige Lösung. Somit haben wir zwei Aspekte für die Qualität eines Constraint-Netzes, nämlich die Konsistenz und die Mehrdeutigkeit. Wie wir später im Abschnitt 3.2.3 sehen werden, liefern beide Gründe zur Abwägung. Im Folgenden führen wir Maße für die Inkonsistenz und die Mehrdeutigkeit ein.

3.2.1. Inkonsistenz

Ein System von Constraints $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ ist inkonsistent falls es keine Lösung nach Definition 3.1.2 besitzt, d.h. falls $\bigcap \mathcal{C}$ leer ist. Ein Maß für die Inkonsistenz sollte also ausdrücken „wie weit“ $\mathcal{C}$ davon entfernt ist eine Lösung zu haben.

Um das Maß zu konstruieren führen wir zunächst eine Metrik für Mengen ein. Sei dazu

$$\tilde{d} : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}^+ \tag{3.3}$$

eine Metrik auf $\mathcal{U}$ (in unseren Beispielen werden wir die Euklidische Distanz verwenden). Wir verallgemeinern die Metrik $\tilde{d}$ auf nicht leere Teilmengen von $\mathcal{U}$ durch

$$d : \mathcal{P}(\mathcal{U}) \setminus \{\emptyset\} \times \mathcal{P}(\mathcal{U}) \setminus \{\emptyset\} \longrightarrow \mathbb{R}^+ \tag{3.4}$$

$$(C, D) \longmapsto d(C, D) := \inf_{c \in C} \inf_{d \in D} \left\{ \tilde{d}(c, d) \right\}. \tag{3.5}$$

Die Funktion d ist bildet keine Metrik auf $\mathcal{P}(\mathcal{U}) \setminus \{\emptyset\}$, da die Dreiecksungleichung im Allgemeinen nicht erfüllt ist. Als Gegenbeispiel nimmt man drei nicht leere Mengen A, B, C wobei sich jeweils $A \cap B \neq \emptyset$, $B \cap C \neq \emptyset$ aber $A \cap C = \emptyset$. Dann ist $d(A, B) + d(B, C) = 0$, aber $d(A, C) > 0$. Für unsre Zwecke ist es jedoch irrelevant. Eine mögliche Alternative bietet an dieser Stelle die *Hausdorff Metrik*

$$d_H(C, D) := \max \left\{ \sup_{c \in C} \inf_{d \in D} \tilde{d}(c, d), \sup_{d \in D} \inf_{c \in C} \tilde{d}(c, d) \right\} \tag{3.6}$$

die alle Axiome einer Metrik auf dem Raum $\mathcal{P}(\mathcal{U}) \setminus \{\emptyset\}$ erfüllt. Zur Vereinfachung schreiben wir für Punkte $p, q \in \mathcal{U}$ und $C \subseteq \mathcal{U}$ auch

$$d(p, q) = d(\{p\}, \{q\}) = \tilde{d}(p, q) \tag{3.7}$$

und

$$d(p, C) = d(\{p\}, C). \tag{3.8}$$

Definition 3.2.1 *(Distanz zum Constraint-Netz)*
Für ein Netz $\mathcal{C} = \{C_1, ..., C_n\}$ von Constraints und ein Punkt $p \in \mathcal{U}$ definieren wir die Distanz von p und $\mathcal{C}$ durch

$$\hat{d}(p, \mathcal{C}) := \sum_{i=1}^{n} d(p, C_i) \tag{3.9}$$

Die Menge aller Punkte $p \in \mathcal{U}$ deren Distanz zum Constraint-Netz $\mathcal{C}$ kleiner oder gleich $s \in \mathbb{R}$ ist wird definiert durch

$$P_{\mathcal{C}}(s) = \{p | d(p, \mathcal{C}) \leq s\} \tag{3.10}$$

Die Abbildung 3.1 veranschaulicht die Mengen $P_{\mathcal{C}}(s)$. Aus dieser Definition folgt direkt:

Korollar 3.2.1 *Eine Menge von Constraints $\mathcal{C}$ ist genau dann konsistent, wenn $P_{\mathcal{C}}(0)$ nicht leer ist.*

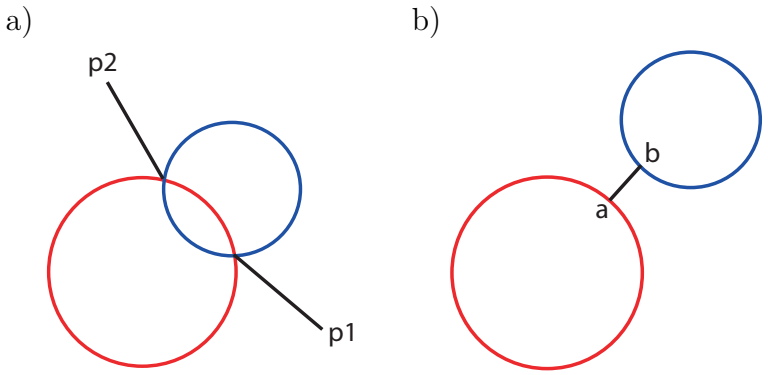


Abbildung 3.2.: *Beispiele für das Inkonsistenz-Maß IK. a) $IK(\mathcal{C}) = 0$. b) $IK(\mathcal{C}) > 0$*

Das führt zu der folgenden Definition des Inkonsistenz-Maßes für Constraint Mengen $\mathcal{C}$:

Definition 3.2.2 *(Inkonsistenz-Maß)*

$$IK(\mathcal{C}) = \min(\{s | P_{\mathcal{C}}(s) \neq \emptyset\}) \tag{3.11}$$

Das Inkonsistenz-Maß lässt sich geometrisch als die kürzeste Distanz interpretieren, die das System $\mathcal{C}$ von Constraints bewegt werden muss um konsistent zu werden:

Korollar 3.2.2

$$IK(\mathcal{C}) = \min_{p} \sum_{i=1}^{n} d(p, C_i) \tag{3.12}$$

Die Abbildung 3.2 illustriert das Maß IK am Beispiel zweier Kreis-Constraints als Constraint-Netz. Wenn beide Kreise sich wie im mittleren Bild schneiden, dann ist das System konsistent mit $IK = 0$. Es kann aber sein, dass die Lösung mehrdeutig ist. Diese Situation werden wir im nächsten Abschnitt näher betrachten. Das rechte Beispiel visualisiert ein inkonsistentes System $IK > 0$. Alle Punkte die auf der Linie zwischen a und b liegen haben dieselbe (Euklidische) Distanz zum Constraint-Netz. Denn, es gilt $IK(\mathcal{C}) = d(a, b)$ und damit ist

$$P_{\mathcal{C}}(IK(\mathcal{C})) = \{p | d(p, C_1) + d(p, C_2) \leq d(a, b) \leq d(p, a) + d(p, b)\} \tag{3.13}$$

genau die Linie zwischen a und b.

Im dem Fall $IK(\mathcal{C}) > 0$ besitzt das System $\mathcal{C}$ keine Lösung. Wenn $IK(\mathcal{C})$ nicht zu groß ist, dann könnte die Menge $P_{\mathcal{C}}(IK(\mathcal{C}))$ als Approximation der Lösung verwendet werden. Wenn aber die Inkonsistenz groß ist, könnte man versuchen eine geeignete konsistente Teilmenge von Constraints zu finden. Dann kann es aber passieren, dass Lösungsmenge und damit die Mehrdeutigkeit der Lösung größer wird. Um diese Situation analysieren zu können, führen wir im Folgenden ein Maß für die Mehrdeutigkeit von Constraint-Netzen ein.

3.2.2. Mehrdeutigkeit

Ohne Constraints ist jedes $p \in \mathcal{U}$ eine mögliche Lösung, falls nur ein Constraint C vorhanden ist, dann sind alle $p \in C$ Kandidaten für die Lösung des Navigationsproblems. Im Unterschied zu anderen CSP besitzt der Roboter eine gewisse Position in der Realität und genau danach suchen wir. Falls die Constraints keine genaue Bestimmung der Position erlauben, ist das Ergebnis mehrdeutig. Eigentlich muss der Roboter das Navigationsproblem lösen um bestimmte Aufgaben erfüllen zu können. Daher kann gewisse Mehrdeutigkeit ohne Bedeutung sein. Es könnte genug sein zu wissen, dass der Roboter sich in einem bestimmten Bereich befindet (z.B. um Abseits bei einem Fußballspiel zu vermeiden).

Auf der anderen Seite, wenn das Constraint-Netz $\mathcal{C}$ inkonsistent ist, dann existieren keine Kandidaten für eine Lösung. Da wir jedoch wissen, dass der Roboter irgendwo in der Welt sein muss, könnten wir nach einer nicht leeren Menge $P_{\mathcal{C}}(s)$ suchen. Solche Mengen mit minimalen Parameter s währen die besten Schätzungen für das Navigationsproblem, insbesondere $P_{\mathcal{C}}(IK(\mathcal{C}))$.

Nun wollen wir eine Definition für das Maß der Mehrdeutigkeit einer beliebigen Menge $P \subseteq \mathcal{U}$ angeben. Dabei gibt es eine Vielzahl von Aspekten, die eine Rolle spielen können. Das Volumen von P könnte ein Maß für die absolute Anzahl möglicher Lösungen sein. Wenn P aus mehreren Komponenten besteht, dann liefert die Anzahl von Komponenten oder die Distanz zwischen den Komponenten ein anderes Maß. Es hängt jedoch sehr stark von der Aufgabe des Roboters und von der Situation ab, welches der Maße besser geeignet ist. Für unsere Studien werden wir beispielhaft nur ein ausgewähltes Maß betrachten. Die einzige Eigenschaft, die dabei für uns eine Rolle spielt, ist die Monotonie. D.h. die Mehrdeutigkeit steigt wenn P größer wird.

Definition 3.2.3 *(Mehrdeutigkeit einer Menge)*
Für eine nicht leere Menge $P \subseteq \mathcal{U}$ definieren wir die Mehrdeutigkeit[2] *durch*

$$Amb(P) = max\{d(p, p') | p, p' \in P\} \tag{3.14}$$

Aus technischen Gründen definieren wir für die leere Menge $Amb(\emptyset) = -1$.

Nach dieser Definition ist die Mehrdeutigkeit $Amb(P)$ genau dann Null, wenn P genau einen Punkt enthält.

3.2.3. Optimale Constraint-Netze

Die Mehrdeutigkeit der Lösung einer CSP kann reduziert werden, indem wir mehr Constraints in Betracht ziehen. Andererseits kann aber dadurch die Inkonsistenz vergrößert werden. Wir müssen also abwägen welche der verfügbaren Constraints

[2]Die Bezeichnung *Amb* kommt von englischer Bezeichnung für Mehrdeutigkeit: *ambiguity*

betrachtet werden sollen. Für die Lokalisierung können viele verschiedene Sensordaten und andere Information benutzt werden, aber aufgrund von Sensorrauschen und Prozessfehlern kann es passieren, dass diese Daten sich zu einem gewissen Grad widersprechen. Es könnte also sinnvoll sein gewisse Inkonsistenz zu akzeptieren um die Mehrdeutigkeit zu reduzieren. Wie wir bereits im Abschnitt 3.2.2 diskutiert haben, hängt es stark von der Aufgabe des Roboters ab, welche Art von Mehrdeutigkeit akzeptabel ist.

Daher ist es notwendig die Constraint-Netze $\mathcal{C}$ zu evaluieren, um entscheiden zu können, ob mehr oder weniger Constraints benutzt werden sollen. Im Folgenden untersuchen wir den Einfluss von Änderungen im Constraint-Netz $\mathcal{C}$ auf die Mehrdeutigkeit und Inkonsistenz.

Offensichtlich wird die Menge der Lösungen mit steigender Anzahl von Constraints kleiner:

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow \bigcap \mathcal{C} \supseteq \bigcap \mathcal{C}' \tag{3.15}$$

Allgemeiner können wir für eine beliebige Zahl s schreiben:

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow P_{\mathcal{C}}(s) \supseteq P_{\mathcal{C}'}(s) \tag{3.16}$$

Für unsere Qualitätsmaße erhalten wir damit

Satz 3.2.1

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow IK(\mathcal{C}) \leq IK(\mathcal{C}') \tag{3.17}$$

$$\mathcal{C} \subseteq \mathcal{C}' \Rightarrow Amb(\bigcap \mathcal{C}) \geq Amb(\bigcap \mathcal{C}') \tag{3.18}$$

Dieser Satz zeigt einen klassischen Kompromiss zwischen der Inkonsistenz und der Mehrdeutigkeit für die Wahl von Constraints. Je größer wir die Teilmenge $\mathcal{C} \subseteq \mathcal{C}'$ wählen, desto kleiner wird die Mehrdeutigkeit der Lösungsmenge und desto größer wird die Inkonsistenz des Systems. Wenn aber die Inkonsistenz größer als Null wird, dann wird die Lösungsmenge leer und die Mehrdeutigkeit gleich -1. Daher liefert uns dieser Satz allein kein sinnvolles Kriterium für die Optimierung.

Für uns haben die Mengen $P_{\mathcal{C}}(IK(\mathcal{C}))$ größere Bedeutung, denn sie können als Ersatz für die Lösungsmenge $\bigcap \mathcal{C}$ verwendet werden in dem Fall, dass das Constraint-Netz $\mathcal{C}$ inkonsistent ist. Auf den ersten Blick scheint es, dass die Mengen $P_{\mathcal{C}}(IK(\mathcal{C}))$ ähnlich der Lösungsmenge in (3.15) ebenfalls kleiner werden würden, wenn $\mathcal{C}$ größer wird. Dann würde auch die Mehrdeutigkeit ähnlich zu (3.18) kleiner werden. Die Mengen $P_{\mathcal{C}}(IK(\mathcal{C}))$ werden jedoch nicht notwendig kleiner wenn das Constraint-Netz größer wird:

Satz 3.2.2 *Es existiert ein Constraint-Netz $\mathcal{C} \subseteq \mathcal{C}'$ so, dass $P_{\mathcal{C}}(IK(\mathcal{C})) \subseteq P_{\mathcal{C}}(IK(\mathcal{C}'))$ und somit $Amb(P_{\mathcal{C}}(IK(\mathcal{C}))) \leq Amb(P_{\mathcal{C}}(IK(\mathcal{C}')))$.*

Als Beispiel betrachten wir die Constraint-Netze $\mathcal{C} = \{\{p\}\}$ und $\mathcal{C}' = \{\{p\}, \{p'\}\}$ für verschiedene Punkte $p, p' \in \mathcal{U}$. Dann gilt $IK(\mathcal{C}) = 0$ und $IK(\mathcal{C}') = d(p, p')$. Damit erhalten wir $P_{\mathcal{C}}(IK(\mathcal{C})) = \{p\} \subseteq \{p, p'\} \subseteq P_{\mathcal{C}'}(IK(\mathcal{C}'))$ was den Satz beweist.

Genauer betrachtet haben wir nach (3.16) natürlich $P_{\mathcal{C}}(s) \supseteq P_{\mathcal{C}'}(s)$ für ein $\mathcal{C} \subseteq \mathcal{C}'$, aber nun können die Werte für s wachsen, wenn wir $s = IK(\mathcal{C})$ für inkonsistente Constraint-Netze benutzen. Für solche Fälle gilt

Satz 3.2.3

$$s \leq s' \Rightarrow P_{\mathcal{C}}(s) \subseteq P_{\mathcal{C}}(s') \tag{3.19}$$

$$s \leq s' \Rightarrow Amb(P_{\mathcal{C}}(s)) \leq Amb(P_{\mathcal{C}}(s')) \tag{3.20}$$

Die Abbildung 3.3 b) zeigt die Werte von $IK(\mathcal{C})$ und $Amb(P_{\mathcal{C}}(IK(\mathcal{C})))$ für alle nicht leeren Teilmengen $\mathcal{C}$ des Constraint-Netzes $\mathcal{C}' = \{C_1, \ldots, C_4\}$ dargestellt in Abbildung 3.3 a). Die Werte für die Folge $\{C_1\} \subset \{C_1, C_2\} \subset \{C_1, C_2, C_3\} \subset \{C_1, C_2, C_3, C_4\}$ liegen nicht auf einer monotonen Geraden im Diagramm, d.h. mehr Constraint können die Mehrdeutigkeit $Amb(P_{\mathcal{C}}(IK(\mathcal{C})))$ vergrößern. Dieses Beispiel veranschaulicht die Tatsache, dass weitere tiefer gehende Untersuchungen notwendig sind um eine optimale Menge von Constraints zu finden.

Am Ende sollten wir noch anmerken, dass die Resultate für die Mehrdeutigkeit nur die Monotonieeigenschaft benutzen, d.h. wenn $P \subseteq P'$, dann $Amb(P) \leq Amb(P')$. Damit sind diese Feststellungen auch für andere Arten der Mehrdeutigkeits-Maße gültig.

3.3. Constraint Propagierung

Im Prinzip können viele Probleme mit Hilfe von Gitter-Basierten Methoden gelöst werden. Wir könnten für jede Zelle testen, ob die Constraints erfüllt sind. Das entspricht einigen bekannten Bayes-Techniken einschließlich Partikelfilter. Die Qualitätsmaße könnten ebenfalls mit Hilfe von Gitter berechnet werden. Allerdings sind diese Techniken viel zu aufwendig für Räume höherer Dimension. Eine alternative Variante bieten Verfahren basierend auf dem Gradientenabstieg.

Eine andere Alternative stellt die Constraint-Propagierung dar. Hier werden die Definitionsbereiche der Variablen durch die Kombination von Constraints sukzessive eingeschränkt. Bekannte Verfahren zur Lösung von Constraintproblemen (wie etwa in [1, 11]) erzeugen damit sukzessive eine monoton fallende Folge von Einschränkungen

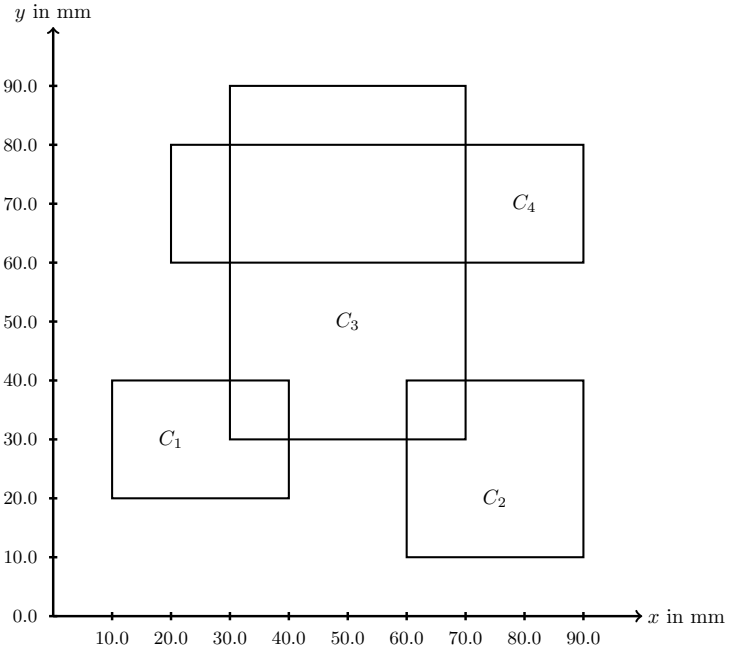
$$\mathcal{U} = D_0 \supseteq D_1 \supseteq D_2 \supseteq \ldots$$

Auf diese Weise wird die Menge alle möglichen Lösungen eingeschränkt.

Nun wollen wir ein Grundschema für die Constraint-Propagierung definieren. Sei dazu folgendes gegeben:

- Ein Constraint-Netz $\mathcal{C} = \{C_1, ..., C_n\}$ über den Variablen $v_1, ..., v_1$ mit dem Wertebereich $\mathcal{U} = Dom(v_1) \times ... \times Dom(v_1)$.

a)



b)

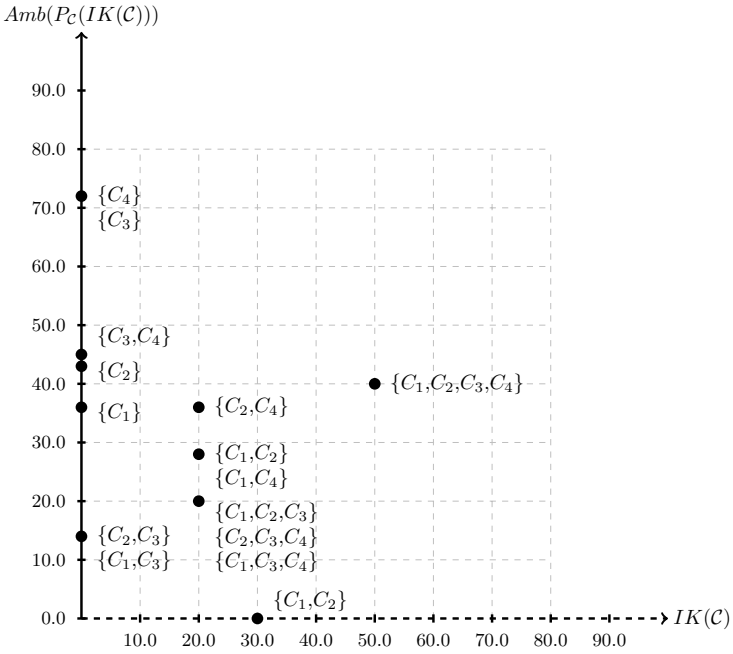


Abbildung 3.3.: *a) Beispiel für ein Constraint-Netz $\mathcal{C}$. b) Inkonsistenz und Mehrdeutigkeits-Maß für potentielle Teilmengen von $\mathcal{C}$.*

- Eine Auswahlfunktion $c : \mathbb{N} \to \mathcal{C}$ wählt in jedem Schritt i ein Constraint der betrachtet werden soll.
- Eine Propagierungsfunktion $d : 2^{\mathcal{U}} \times \mathcal{C} \to 2^{\mathcal{U}}$ für die Propagierung von Constraints die bezüglich der Mengen Inklusion monoton fallend im ersten Argument ist, d.h. $d(D, C) \subseteq D$.
- Eine Stopp-Funktion $t : \mathbb{N} \to \{true, false\}$.

Dann können wir das Schema wie folgt angeben:

Algorithmus 1 : Basic Scheme for Constraint Propagation, BSCP

Input : $D_0 := \mathcal{U}$, $i := 1$
Result : restriction D
1 Step(i): $D_i := d(D_{i-1}, c(i))$.
2 If $t(i) = true$: Stop.
3 Otherwise $i := i + 1$, continue with Step(i)

Definition 3.3.1 *(Basic Scheme for Constraint Propagation, BSCP[3]) Wir nennen jeden Algorithmus, der diesem Schema entspricht einen BSCP-Algorithmus.*

Durch Einschränkungen wird der Suchraum möglicher Lösungen verkleinert. Wenn jedoch die Einschränkungen zu stark sind, dann können Lösungen verloren gehen. In solchen Fällen kann Backtracking eingesetzt werden.

Um das Schema einfach zu halten, hängen die Funktionen c und t nur von dem Zeitschritt ab. Eine einfache Strategie für c ist alle Constraint in $\mathcal{C}$ nacheinander durchzugehen oder zufällig auszuwählen.

Ein besseres Stopp-Kriterium t könnte die Änderungen der Mengen D_i miteinbeziehen, z.B. bricht der Algorithmus ab, wenn die Einschränkungen D_i sich nicht mehr ändern. Zu beachten ist hier, dass die Folge der Einschränkungen nicht zu konvergieren braucht, wenn $D_i = D_{i-1}$ für ein i gilt. Um die Konvergenz zu sichern müssen mehrere Schritte betrachtet werden. Im Allgemeinen braucht die Folge $D_0, D_1, D_2, \ldots$ überhaupt nicht zu konvergieren. Als einfaches Beispiel dafür betrachten wir das folgende CSP mit Variablen $V = v_1, v_2, Dom(v_1) = Dom(v_2) = [0, 1], \mathcal{U} := Dom(v_1) \times Dom(v_2)$ und Constraints

$$C_1 := \{[a, b] | a = b\} \subseteq \mathcal{U} \tag{3.21}$$
$$C_2 := \{[a, b] | a = 2b\} \subseteq \mathcal{U}. \tag{3.22}$$

Offensichtlich besitzt dieses CSP die globale Lösung $[0, 0]$. Nun lösen wir das Problem mit Hilfe der Constraint-Propagierung. Dazu starten wir mit der initialen Einschränkung $D_0 = [0, 1] \times [0, 1]$. Als Propagierungsfunktion verwenden wir den

[3] *Grundschema für die Constraint-Propagierung* die Bezeichnung wurde bewusst Englisch gehalten um den Bezug zu den entsprechenden Publikationen aufrecht zu erhalten.

Mengendurchschnitt. Bei der Propagierung wird in jedem Schritt die aktuelle Einschränkung mit einem der Constraints geschnitten. Das führt dazu, dass eines der Intervalle halbiert wird, wodurch eine unendliche Folge von Einschränkungen entsteht:

$$D_1 := C_2 \cap D_0 = [0,1] \times [0,0.5] \tag{3.23}$$
$$D_2 := C_1 \cap D_1 = [0,0.5] \times [0,0.5] \tag{3.24}$$
$$...$$

Bei Lokalisierungsproblemen mit einfachen Constraints ist es möglich die Lösung direkt zu berechnen:

Korollar 3.3.1 *Sei die Propagierungsfunktion d gegeben durch $d(D,C) := D \cap C$ für alle $D \subseteq \mathcal{U}$ und alle $C \in \mathcal{C}$. Dann wird die Folge der Einschränkungen nach $n = card(\mathcal{C})$ Schritten stationär mit dem korrekten Ergebnis $D_n = \bigcap \mathcal{C}$.*

Üblicherweise werden bei den CSPs nur einige, aber nicht notwendig alle Lösungen gesucht. In diesem Fall braucht die Einschränkungsfunktion d nicht alle Lösungen zu berücksichtigen, d.h. sie braucht nicht *konservativ* gemäß der nachfolgenden Definition 3.3.3 zu sein.

Eine häufig verwendete Bedingung ist die lokale Konsistenz:

Definition 3.3.2 *(Lokal konsistente Propagierungsfunktion)*

1. *Eine Einschränkung D heißt* **lokal konsistent bezüglich eines Constraints** *C wenn*

   $$\forall d = [d_1, ..., d_k] \in D \quad \forall i = 1, ..., k \quad \exists d' = [d'_1, ..., d'_k] \in D \cap C \ : d_i = d'_i$$

   d.h. wenn jeder Wert einer Variable in einer Belegung aus D zu einer Belegung in D vervollständigt werden kann, die C erfüllt.

2. *Eine Propagierungsfunktion $d : 2^{\mathcal{U}} \times \mathcal{C} \to 2^{\mathcal{U}}$ heißt* **lokal konsistent**, *wenn für alle D und C die resultierende Einschränkung $d(D,C)$ lokal konsistent bezüglich C ist.*

3. *Die* **maximale lokal konsistente** *Propagierungsfunktion $d_{maxlc} : 2^{\mathcal{U}} \times \mathcal{C} \to 2^{\mathcal{U}}$ ist definiert durch $d_{maxlc}(D,C) := Max\{d(D,C) | d \text{ is locally consistent}\}$.*

Die Suche nach einer Lösung ist in einem kleineren Raum einfacher, daher wird die Constraint Propagierung oft mit restriktiveren Propagierungsfunktionen als mit d_{maxlc} durchgeführt. Falls keine Lösung gefunden wird, kann Backtracking zu anderen Einschränkungen eingesetzt werden.

Es ist anders bei den Lokalisierungsaufgaben: Hier wollen wir einen Überblick über alle möglichen Positionen haben. Weiterhin, wenn ein klassisches CSP inkonsistent ist, dann hat es keine Lösung. Ein Lokalisierungsproblem dagegen besitzt in

der Realität immer eine Lösung (die realen Positionen betrachteter Objekte). Die Inkonsistenz kann z.B. durch verrauschte Sensordaten verursacht werden. In diesem Fall müssen einige Constraints aufgeweicht oder erweitert werden. Das kann während des Propagierungsprozesses dadurch erfolgen, dass größere als die maximale lokal konsistenten Einschränkungen verwendet werden.

Definition 3.3.3 *(Konservative Propagierungsfunktion)*
Eine Propagierungsfunktion $d : 2^{\mathcal{U}} \times \mathcal{C} \to 2^{\mathcal{U}}$ *heißt* **konservativ** *wenn* $D \cap C \subseteq d(D, C)$ *für alle* D *und* C*.*

Man beachte, dass die maximale lokal konsistente Propagierungsfunktion d_{maxlc} konservativ ist. Damit erhalten wir:

Satz 3.3.1 *Sei die Propagierungsfunktion* d *konservativ.*

1. *Dann gilt für alle Einschränkungen* $D_i : \bigcap \mathcal{C} \subseteq D_i$*.*
2. *Wenn eine der Einschränkungen* D_i *leer ist, dann existiert keine Lösung, d.h.* $\bigcap \mathcal{C} = \emptyset$*.*

Falls keine Lösung gefunden werden kann, dann ist das Constraint-Netz inkonsistent. Es gibt unterschiedliche Strategien damit umzugehen:

- Vergrößerung einer Constraints aus $\mathcal{C}$,
- Betrachtung einer konsistenten Teilmenge von $\mathcal{C}$,
- Berechnung einer am besten zu $\mathcal{C}$ passenden Hypothese.

Wir haben diese Möglichkeiten bereits im Abschnitt 3.2.3 ausführlich diskutiert.

3.4. Intervall Constraints

Im Fall numerischer Constraints kann die Berechnung von Einschränkungen und damit die Propagierung von Constraints sehr aufwendig werden, wenn die Constraints komplizierte Formen annehmen. Zur Vereinfachung der Propagierung werden oft mehrdimensionale Intervalle/Polyeder verwendet um die Constraints und die Einschränkungen zu approximieren.

Ein *k-dimensionales Intervall* ist eine Menge

$$I = [a, b] := \{x | a \leq x \leq b\} \subseteq \mathcal{U} \tag{3.25}$$

wobei $a, b \in \mathcal{U}$ und die Relation $\leq$ komponentenweise definiert ist. Die Menge aller Intervalle im Universum $\mathcal{U}$ bezeichnen wir mit $\mathcal{I}$.

Bei der Propagierung können die Constraints mit Intervallen geschnitten und die kleinste Intervall-Hülle des Schnitts als konservatives Ergebnis benutzt werden. In

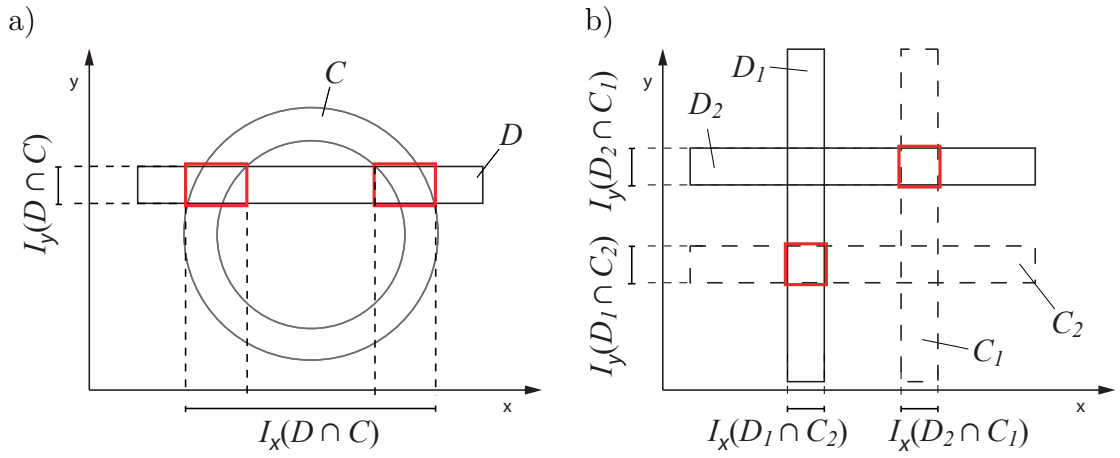


Abbildung 3.4.: *Propagierung eines Constraints mit einem Intervall D für a) einen kreisförmigen Constraint C und b) einen Intervall Constraint C. Das* Projektionsintervall *bezüglich $C \cap D$ wird illustriert.*

der Abbildung 3.4 werden einige Beispiele der Schnitte von Constraints mit Intervallen illustriert.

Definition 3.4.1 *(Intervall Propagierung)*

1. *Eine Propagierungsfunktion d heißt* **Intervall Propagierungsfunktion** *wenn die Werte von d stets Intervalle sind, d.h. $d : 2^{\mathcal{U}} \times \mathcal{C} \to \mathcal{I} \subset 2^{\mathcal{U}}$.*

2. *Die* **minimale konservative** *Intervall Propagierungsfunktion $d_{minc} : 2^{\mathcal{U}} \times \mathcal{C} \to \mathcal{I}$ ist definiert durch*

   $d_{minc}(D, C) := \bigcap_{I \in \mathcal{I},\ D \cap C \subseteq I} I = \min\{I | I \in \mathcal{I} \wedge D \cap C \subseteq I)\}$

   für alle D und C.

Das Ergebnis minimal konservativer Propagierungsfunktionen kann mit Hilfe von Projektionen berechnet werden.

Definition 3.4.2 *(Projektionsintervall)*
Der (eindimensionale) **Projektionsintervall bezüglich einer Menge** *$M \subseteq \mathcal{U}$ für eine Variable v ist definiert als das kleinste Intervall das die Projektion $\pi_v(M)$ von M auf die Variable v beinhaltet, d.h.*

$$I_v(M) := \bigcap_{\substack{I \in \mathcal{I} \\ \pi_v(M) \subseteq I}} I = \min\{I | I \subseteq \mathbb{R} \wedge \pi_v(M) \subseteq I\}$$

Es kann berechnet werden als Intervall $I = [a, b]$ mit $a := \min(\pi_v(M))$ and $b := \max(\pi_v(M))$. Das Minimum und Maximum wird dabei gegebenenfalls komponentenweise gebildet.

Sowohl die maximale lokale Konsistenz, als auch der minimale Konservatismus führen zu demselben Resultat und können beide mit Hilfe von Projektionen berechnet werden (siehe Abbildung 3.4):

Satz 3.4.1

1. $d_{maxlc}(D, C) = d_{minc}(D, C)$

2. $d_{minc}(D, C) = I_{v_1}(D \cap C) \times .... \times I_{v_k}(D \cap C).$

Die lokale Konsistenz ist der klassische Ansatz um einige Lösungen zu finden. Der Ansatz mit konservativen Intervallen ist hingegen für die Lokalisierungsaufgaben besser geeignet, denn es lässt sich während der Propagierung hinsichtlich Vergrößerung von Constraints modifizieren, sodass Inkonsistenz vermieden werden kann.

Der folgende einfache und praktikable Algorithmus wird für die Propagierung von Constraints verwendet. Die Stopp-Bedingung vergleicht den Fortschritt nach jedem einzelnen Schritt. Da die Folge der Einschränkungen nicht konvergieren muss, wird ein Zeitlimit als zusätzliches Abbruchkriterium gesetzt. Man sollte beachten, dass der Schrittzähler s nicht mit den Schritten i des Grundschemas BSCP übereinstimmt (es könnte aber entsprechend angepasst werden).

Algorithmus 2 : Constraint Propagation with Minimal Conservative Intervals, MCI-algorithm

Input : Constraint-Netz $\mathcal{C} = \{C_1, ..., C_n\}$ mit Variablen $\mathcal{V} = \{v_1, ..., v_k\}$ über dem Definitionsbereich $\mathcal{U}$ und eine Zeitgrenze T
Data : $D \leftarrow \mathcal{U}$, $s \leftarrow 1$, $D_{old} \leftarrow \emptyset$
Result : minimales konservatives k-dimensionales Intervall D

```
1  while s < T  &  D ≠ D_old do
2  |  D_old ← D;
3  |  foreach C ∈ C do
4  |  |  foreach v ∈ V do
5  |  |  |  D(v) ← I_v(D ∩ C);
6  |  |  end
7  |  |  D ← D(v_1) × ··· × D(v_n);
8  |  end
9  |  s ← s + 1;
10 end
```

Der Algorithmus arbeitet auf einzelnen Intervallen. Bei näherer Betrachtung möglicher Schnitte von Constraints sehen wir sofort, dass in einigen Fällen die Approximation durch einzelne Intervalle sehr ungenau sein kann. Die Abbildung 3.4 a) illustriert einen Schnitt eines Kreisrings mit einem Intervall. Offensichtlich können in solchen Fällen die Mengen $D \cap C$ viel besser durch mehrere Intervalle approximiert werden, wie in der Abbildung 3.4 b) veranschaulicht.

Der Algorithmus lässt sich leicht derart erweitern, dass er auf Mengen von Intervallen arbeitet. Darauf gehen wir im nächsten Abschnitt genauer ein.

3.4.1. Intervallvereinigung

Wie bereits angesprochen können komplizierte Constraints oft nur sehr ungenau durch Intervalle approximiert bzw. eingeschränkt werden. Ein einfaches Beispiel wird in der Abbildung 3.4 illustriert. Hier wird ein Ring-Constraint mit einem Intervall propagiert, dabei entsteht eine recht ungenaue Annäherung. Um eine bessere Approximation der Constraints durch Intervalle zu erhalten, erweitern wir unsere Betrachtungen auf Vereinigung von Intervallen. Dabei wird ein Constraint durch eine Menge von Intervallen dargestellt.

Zunächst versuchen wir diese Idee intuitiv zu erklären. Sei dazu eine Menge $C = \{I_1, ..., I_n\} \subset \mathcal{I}$ von Intervallen gegeben. Nun wollen wir C als einzelnes Constraint betrachten, d.h. eine Belegung $\beta \in \mathcal{U}$ erfüllt C genau dann, wenn $\beta \in \bigcup_{j=1,...,n} I_j$. Wir wollen also einerseits C als Menge von Intervallen betrachten, andererseits soll aber C auch ein Constraint (also eine Teilmenge von $\mathcal{U}$) repräsentieren. Unter diesem Gesichtspunkt stellen die markierten Intervalle in der Abbildung 3.4 zusammen ein Constraint dar.

Um diesen Sachverhalt zu formalisieren definieren wir zunächst die Menge

$$\mathcal{I}^{\mathcal{U}} := \{\iota | \iota = \bigcup_{I \in C} I, C \subset \mathcal{I}, |I| < \infty\} \subset 2^{\mathcal{U}} \tag{3.26}$$

Anders formuliert besteht die Menge $\mathcal{I}^{\mathcal{U}}$ aus solchen Teilmengen (Constraints) von $\mathcal{U}$ für die eine endliche Überdeckung durch Intervalle aus $\mathcal{I}$ existiert. Eine solche Überdeckung ist offensichtlich nicht eindeutig, da wir jedes Intervall als Vereinigung zweier Teilintervalle darstellen können. Die einzelnen Intervalle einer Vereinigung müssen auch nicht notwendig disjunkt sein. Bei der Algorithmischen Umsetzung wollen wir jedoch auf die Intervallstruktur einer Intervallvereinigung zugreifen. Also definieren wir für ein $\iota \in \mathcal{I}^{\mathcal{U}}$ die Klasse

$$[\iota] := \{I | \iota = \bigcup_{I \in C} C, I \subset \mathcal{I}, |I| < \infty\} \tag{3.27}$$

aller endlichen Überdeckungen von ι durch Intervalle. Die nachfolgenden Betrachtungen sind unabhängig von dem gewählten Repräsentanten, daher können wir ohne Beschränkung der Allgemeinheit annehmen, dass $[\iota]$ eine *fest gewählte* Intervallüberdeckung von ι darstellt.

Die Propagierung von Intervallen ist algorithmisch sehr einfach realisierbar, da die Schnitte von Intervallen einfach berechnet werden können. Diese Eigenschaft lässt sich auch auf Vereinigungen von Intervallen übertragen. Für $\iota_1, \iota_2 \in \mathcal{I}^{\mathcal{U}}$ gilt

$$\iota_1 \cap \iota_2 = \left(\bigcup_{I \in [\iota_1]} I \right) \cap \left(\bigcup_{J \in [\iota_2]} J \right) \bigcup_{I \in [\iota_1],\, J \in [\iota_2]} (I \cap J) \tag{3.28}$$

Da Schnitte von Intervallen wieder Intervalle sind gilt $\iota_1 \cap \iota_2 \in \mathcal{I}^{\mathcal{U}}$. Dieses Konzept

können wir auf beliebige Propagierungsfunktionen d verallgemeinern, sofern d sich im ersten Argument *distributiv* bezüglich der Vereinigung verhält:

$$d(D, \iota) = \bigcup_{I \in [\iota]} d(D, I) \tag{3.29}$$

wobei $D \in 2^{\mathcal{U}}$ eine Einschränkung und $\iota \in \mathcal{I}^{\mathcal{U}}$ eine Intervallvereinigung ist.

Der Algorithmus 2 kann ohne Weiteres auf die Intervallvereinigungen erweitert werden.

3.4.2. Interpolation von Constraints (Soft-Cut)

Verrauschte Sensordaten können zu Inkonsistenzen führen. Wenn dieser Fall eintritt, so können die Constraints nicht mehr propagiert werden, obwohl die Inkonsistenzen eventuell nicht so groß sind. In diesem Abschnitt stellen wir eine alternative konservative Propagierungsfunktion vor, die trotz der Inkonsistenz eine Propagierung von Constraints erlaubt.

Für zwei beliebige Teilmengen $A, B \subseteq \mathcal{U}$ und ein $\lambda \in [0, 1]$ können wir die lineare Interpolation zwischen A und B definieren:

$$d_t(A, B) = \{x | x = a \cdot (1 - \lambda) + b \cdot \lambda, a \in A, b \in B\} \subseteq \mathcal{U} \tag{3.30}$$

Die Funktion $d_\lambda : 2^{\mathcal{U}} \times 2^{\mathcal{U}} \to 2^{\mathcal{U}}$ ist offensichtlich konservativ, denn es gilt

$$x \in A \cap B \Rightarrow x = x \cdot (1 - \lambda) + x \cdot \lambda \in d_\lambda(A, B) \tag{3.31}$$

und damit also $A \cap B \subseteq d_t(A, B)$. Im Fall von Intervallen lässt sich die Funktion d_λ sehr einfach berechnen. Seien $I, J \in \mathcal{I}$ zwei Intervalle und $\lambda \in [0, 1]$ fixiert, dann gilt

$$d_\lambda(I, J) = I \cdot (1 - \lambda) + J \cdot \lambda \tag{3.32}$$

Die Funktion d_λ verhält sich distributiv gegenüber der Vereinigung $\cup$ von Mengen, d.h. für beliebige Mengen $A, B, C \subseteq \mathcal{U}$ gilt

$$d_\lambda(A \cup B, C) = d_t(A, C) \cup d_\lambda(B, C). \tag{3.33}$$

Wegen der Reflexivität von d_t gilt ebenfalls die Analogie für das zweite Argument. Das lässt sich wie folgt elementar nachweisen:

$$\begin{aligned}
d_t(A \cup B, C) &= \{x | x = a \cdot (1 - \lambda) + c \cdot \lambda, a \in A \cup B, c \in C\} && (3.34)\\
&= \{x | x = a \cdot (1 - \lambda) + c \cdot \lambda, a \in A \vee a \in B, c \in C\} && (3.35)\\
&= \{x | x = a \cdot (1 - \lambda) + c \cdot \lambda, a \in A, c \in C\} \cup && (3.36)\\
&\quad \{x | x = a \cdot (1 - \lambda) + c \cdot \lambda, a \in B, c \in C\} && (3.37)\\
&= d_\lambda(A, C) \cup d_\lambda(B, C) && (3.38)
\end{aligned}$$

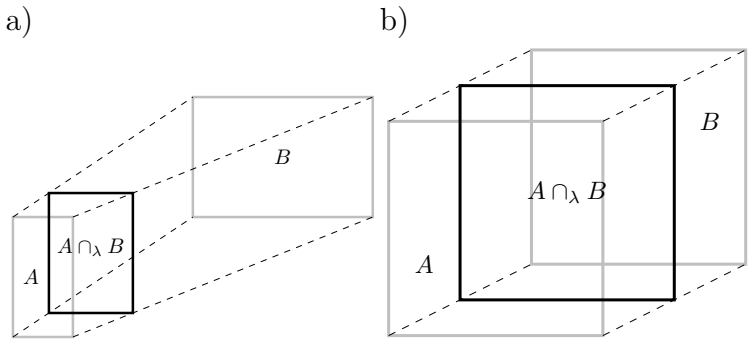


Abbildung 3.5.: *Weiche Schnitte (*Soft-Cut*) von Constraints am Beispiel von Intervallen. Die grauen Intervalle A und B werden miteinander propagiert. Die schwarzen Boxen stellen jeweils das Resultat $A \cap_\lambda B$ dar. Im Beispiel links wird die Interpolation mit $\lambda = 0.2$ und rechts mit $\lambda = 0.5$ durchgeführt. Am rechten Beispiel sieht man, dass der Schnitt der Intervalle A und B im weichen Schnitt enthalten ist, d.h. Soft-Cut ist konservativ.*

Damit können wir die Funktion d_λ als einen „weichen“ Schnitt-Operator (*Soft-Cut*) $\cap_\lambda$ interpretieren.

Die Soft-Cuts von Intervallvereinigungen lassen sich damit auch komponentenweise berechnen. Das lässt sich analog zum klassischen Schnitt $\cap$, wie in der Gleichung 3.28 gezeigt, herleiten.

3.4.3. Intervall-Hülle

Beim näheren Hinschauen lässt sich einfach feststellen, dass sowohl der Mengendurchschnitt als Propagierungsfunktion als auch der *Soft-Cut* im schlechtesten Fall quadratisch bezüglich der Anzahl zusammenhängender Komponenten sind. Das bedeutet, dass die Anzahl der einzelnen Intervalle beim Propagieren von Intervallvereinigungen quadratisch ansteigt. Das kann direkt aus der Gleichung 3.28 abgeleitet werden. Oft entstehen dabei auch Intervalle die ineinander enthalten sind, oder sich nur wenig unterscheiden. Die Abbildung 3.6 veranschaulicht einige Beispiele solcher Konstellationen. In solchen Fällen kann es Sinn machen „ähnliche“ Constraints zusammenzufassen. Im Folgenden überlegen wir uns wie die Constraints zusammengefasst werden können und welche Kriterien für die Ähnlichkeit benutzt werden können.

Definition 3.4.3 *(Intervall-Hülle)*

Seien $A, B \in \mathcal{I}$ zwei Intervalle. Dann definieren wir die Intervall-Hülle *von A und B durch als das minimale Intervall bezüglich Inklusion, dass A und B beinhaltet, d.h.*

$$H_I(A,B) := \bigcap_{\substack{D \in \mathcal{I} \\ A,B \subset D}} D$$

Diese Definition lässt sich ohne weiteres auf eine Menge von Intervalle erweitern. Sei dazu eine Menge von Intervallen gegeben

$$\mathcal{C} \subseteq \mathcal{I}. \tag{3.39}$$

Wir definieren die *Intervall-Hülle* $H_I(\mathcal{C})$ von $\mathcal{C}$ analog zur obigen Definition wie folgt

$$H_I(\mathcal{C}) := \bigcap_{\substack{D \in \mathcal{I} \\ \forall C \in \mathcal{C} : C \subset D}} D \tag{3.40}$$

Es gibt viele Möglichkeiten Ähnlichkeit zweier Menge zu definieren, z.B. über die Fläche, Form, etc.. Welche Maße für die Ähnlichkeit sinnvoll sind hängt dabei stark von der Anwendung ab. Wie wir bereits im Abschnitt 3.2 diskutiert haben, kann das Inkonsistenz-Maß als untere und das Mehrdeutigkeits-Maß als obere Optimalitätsbedingung für Constraints angesehen werden. Da die Bildung der Intervall-Hülle monoton wachsend ist, brauchen wir eine obere Bedingung dafür, welche Constraints durch eine Hülle vereinigt werden sollen. Da die Intervall-Hülle immer größer als die Vereinigung einzelner Mengen ist, folgt mit der Monotonie des Mehrdeutigkeits-Maßes:

Korollar 3.4.1

$$Amb(H_I(A,B)) \geq Amb(A \cup B)$$

Wir können also auf der Basis eines Mehrdeutigkeits-Maßes Amb eine allgemeine Bedingung formulieren:

$$Amb(H_I(A,B)) \leq \delta \cdot Amb(A \cup B) \tag{3.41}$$

für ein gewähltes $\delta \geq 1$. Der Parameter δ kann als der „maximale Vergrößerungsfaktor“ gedeutet werden. Wenn z.B. $\delta = 2$ gewählt wird, so ist die obere Ungleichung nur in den Fällen erfüllt, in denen sich die Mehrdeutigkeit maximal verdoppelt. Die

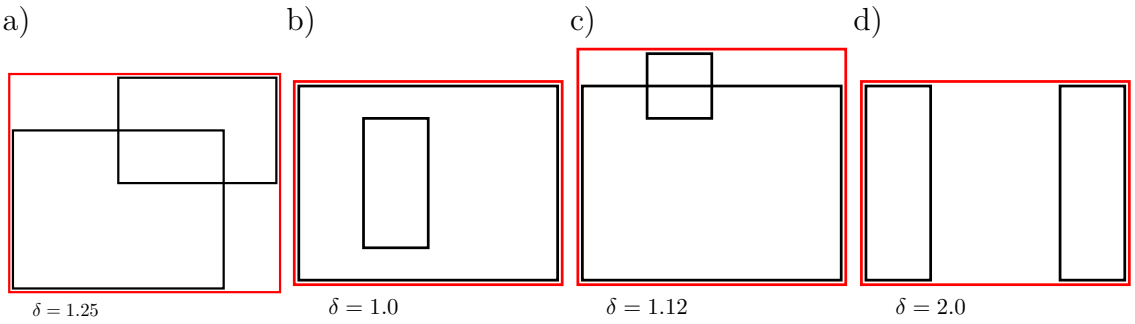


Abbildung 3.6.: *Diese Abbildung veranschaulicht einige mögliche Konstellationen zweier Intervallconstraints und die entsprechende Intervall-Hüllen. Für jede Konstellation wurde der Vergrößerungsfaktor* $\delta = \frac{Amb(A \cup B)}{Amb(H_I(A,B))}$ *berechnet, wobei als Maß Amb für Mehrdeutigkeit der Flächeninhalt verwendet wurde.*

Abbildung 3.6 illustriert einige Beispiele für die Bildung einer Intervall-Hülle. Hier wurde der Flächeninhalt als Maß für die Mehrdeutigkeit verwendet.

Algorithmus 3 : Intervall-Hülle

Input : $\mathcal{C} \subset \mathcal{I}$ //endliche Menge von Intervall-Constraints
Result : $\hat{\mathcal{C}} \subset \mathcal{I}$ //mit $k \leq n$ und $\bigcup \mathcal{C} \subseteq \bigcup \hat{\mathcal{C}}$
Data : $changed \leftarrow false$

```
1  while changed = true do
2  |   C ∈ C;
3  |   C ← C \ {C};
4  |   changed ← false;
5  |   foreach D ∈ C do
6  |   |   if Amb(H_I(C, D)) ≤ δ · Amb(C ∪ D) then
7  |   |   |   C ← H_I(C, D) ;
8  |   |   |   C ← C \ {D};
9  |   |   |   changed ← true;
10 |   |   end
11 |   end
12 |   C ← C ∪ {C};
13 end
14 return C_t^B
```

Der maximal zulässige Vergrößerungsfaktor δ geht dabei als Parameter in den Algorithmus ein. Die Abbildung 3.7 illustriert die berechnete Intervall-Hülle für das Beispiel der Lokalisierung.

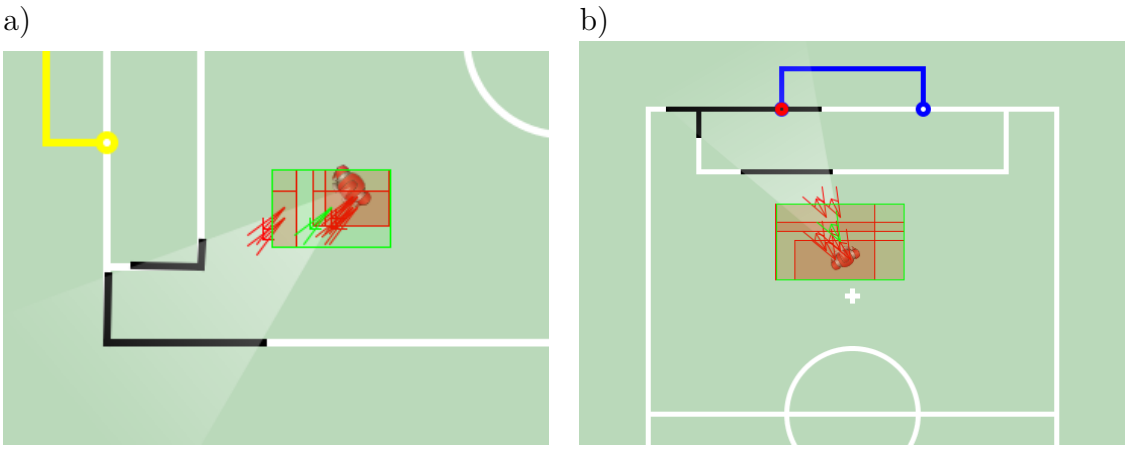


Abbildung 3.7.: *Nach dem Schnitt von Constraints kann die modellierte Position aus mehreren Constraints bestehen (kleine Boxen). Die Constraint-Hülle berechnet mit dem Algorithmus 3 vereinigt redundante Constraints zu einem einzigen (umfassende Box).*

Die Bildung der Intervall-Hülle kann als eine erweiterte Form der Vereinigung von Mengen angesehen werden. Wir schreiben auch

$$A \cup_I B := H_I(A, B) \tag{3.42}$$

In dem Fall, dass die Vereinigung $A \cup B \in \mathcal{I}$ ein Intervall ist, gilt $A \cup B = H_I(A, B)$. Im Gegensatz zu der klassischen Vereinigung ist die Menge der Intervalle $\mathcal{I}$ bezüglich $\cup_I$ abgeschlossen.

Eine andere Variante der Optimierung wäre die Verkleinerung von Constraints. An dieser Stelle könnte eine analoge Bedingung auf der Basis eines Maßes für die Inkonsistenz definiert werden.

3.5. Zusammenfassung

Die Modellierung mit Constraints umfasst eine Vielzahl von Techniken. Wie bei jedem Modellierungsansatz sind die zentralen Fragen: *„Wie wird das Wissen repräsentiert?“* und *„Wie kann neues Wissen in das bestehende Modell integriert werden?“*.

Wir haben die Constraints als Teilmengen des gesamten Zustandraumes (des Universums) definiert. Damit lassen sich beliebige Einschränkungen darstellen. Komplexe Constraints können aber zu einem sehr hohen Berechnungsaufwand führen. Um die Komplexität zu reduzieren haben wir die Approximation von Constraints durch Intervalle und durch Vereinigungen von Intervallen eingeführt, was eine einfache algorithmische Umsetzung ermöglicht.

Für die Integration von Constraints haben wir die Constraint-Propagierung mit konservativen Intervallen vorgestellt. Dabei werden die Schnitte von Constraints

durch konservative Intervalle angenähert. Das vereinfacht das Bilden der Einschränkungen und sichert, dass keine globale Lösungen verloren gehen.

Ein besonderes Problem bei der Modellierung mit Constraints stellen die Inkonsistenzen dar. Wenn die Daten verrauscht sind, kann es dazu führen, dass keine Lösung des Problems gefunden werden kann. Wenn aber die Constraints zu schwach sind, dann wird die Lösung mehrdeutig und verliert damit an Aussagekraft. Um hier ein Optimum zu finden haben wir Maße für die Inkonsistenz und Mehrdeutigkeit der Constraint-Netze eingeführt.

Eine Alternative zum Mengendurchschnitt als Propagierungsfunktion stellt die Interpolation (Soft-Cut) von Constraints dar. Soft-Cut ist robuster gegenüber Inkonsistenzen, aber nicht so restriktiv wie der Mengendurchschnitt, wodurch die Lösungsmenge größer wird. Um die Berechnungskomplexität zu reduzieren wurde ein Hüllenoperator konstruiert, der die Anzahl von Teilconstraints einer Constraint-Vereinigung reduzieren kann.

Im Folgenden stellen wir eine konkrete Implementierung einer auf Constraints basierten Lokalisierung vor.

4. Lokalisierung mit Constraints

In diesem Abschnitt stellen wir eine Umsetzung des Constraint basierten Modellierung für die Aufgabe der Lokalisierung eines mobilen Roboters vor und untersuchen die praktische Realisierbarkeit anhand einer konkreten Implementierung für den Einsatz im RoboCup.

Im RoboCup wird die Anzahl künstlicher Landmarken jedes Jahr reduziert. Es müssen also andere sensorische Informationen, wie etwa die Feldlinien, stärker verwendet werden. Die Linieninformation resultiert in sehr komplizierten Verteilungen, wenn sie stochastisch modelliert wird, die nur schwer durch Gauss-Verteilungen oder Partikel dargestellt werden kann. Unter diesem Gesichtspunkt scheint der Einsatz der Constraint Techniken sehr vielversprechend zu sein. Denn im konkreten Fall ist die Beschreibung der durch die Linien definierten Constraints tatsächlich sehr einfach, wie wir gleich am Anfang des Abschnittes sehen werden.

Als erstes untersuchen wir auf eine allgemeine Weise, wie die Sensordaten des Roboters dazu verwendet werden können Constraints aufzubauen und stellen einige konkrete Beispiele vor. Danach werden einige konkrete Umsetzungen des BSCP-Schemas vorgestellt und der Einfluss von Inkonsistenz diskutiert. Am Ende analysieren wir die Leistung der Lokalisierung in mehreren Experimenten auf verschiedenen Roboter-Plattformen.

4.1. Perzept-Constraints

Die zentrale Frage in diesem Abschnitt ist wie die Sensordaten des Roboters dazu verwendet werden können Constraints zu erzeugen. Wir bezeichnen solche Constraints auch als *Perzept-Constraints*[1].

Einer der wichtigsten Sensoren eines Roboters ist die Kamera. Mit Hilfe geeigneter Bildverarbeitung und der Information über die Lage des Roboters im Raum (vgl. Abschnitt 2) können Objekte in der Umgebung des Roboters (wie etwa Linien oder Tore) erkannt und ihre Eigenschaften wie die Farbe, Größe oder relative Lage zum Roboter bestimmt werden. Wir können also diese berechneten Eigenschaften der Objekte als Daten gewisser abstrakter Sensoren betrachten. Damit können wir auch von *Sensordaten* und *Messungen* sprechen. Insbesondere brauchen wir dabei nicht zwischen Kamera und anderen Sensoren zu unterscheiden. So gibt es dann z.B. einen Sensor der den durchschnittlichen Farbton des gesehenen Balls, einen Winkel

[1] Als *Perzept* bezeichnet man ein Wahrnehmungserlebnis.

zwischen beiden Torpfosten, die Entfernung zu einem anderen Roboter bestimmt oder die Information darüber liefert, ob ein bestimmtes Objekt gesehen wurde oder nicht.

Seien $v_1, v_2, ..., v_m$ verschiedene Variablen, die die Messungen verschiedener Sensoren S_i repräsentieren mit ihren Definitionsbereichen $Dom(v_i) \subseteq \mathbb{R}$. Die (geometrischen) Zusammenhänge zur Zeit t zwischen den Variablen können in der allgemeinen Form

$$f_t(v_1, v_2, ..., v_m) = 0 \tag{4.1}$$

angegeben werden. Ein Sensor S_i misst/liefert eine Belegung $\hat{w}_i^t \in Dom(v_i)$ für die Größe v_i. Den exakten/tatsächlichen Wert von v_i zur Zeit t bezeichnen wir mit w_i^t.

Das liefert uns folgenden allgemeinen Ansatz zur Konstruktion eines Constraints basierend auf einer Messung $\hat{w}_i^t \in Dom(v_i)$ des Sensors S_i:

$$C = C(f_t, i) := \{\beta \in \mathcal{U} | f_t(\beta(v_1), ..., \beta(v_m)) = 0 \wedge \beta(v_i) = \hat{w}_i^t\}. \tag{4.2}$$

Da die Messungen meistens fehlerbehaftet sind, sollten die Constraints nicht exakt definiert werden. Um die Fehler in der Messung widerzuspiegeln lassen wir eine Abweichung $\varepsilon > 0$ zu. Das angepasste Constraint hat dann die Form

$$\hat{C} = \hat{C}(f_t, i, \varepsilon) := \{\beta \in \mathcal{U} | f_t(\beta(v_1), ..., \beta(v_m)) = 0 \wedge \beta(v_i) \in [\hat{w}_i^t - \varepsilon, \hat{w}_i^t + \varepsilon]\}. \tag{4.3}$$

Eigentlich stellt dieses Constraint einen Schnitt der Menge der Punkte in $\mathcal{U}$ die den Zusammenhang f_t erfüllen und der Menge der Punkte, deren Projektion auf die i-te Variable im Toleranzbereich $[\hat{w}_i^t - \varepsilon, \hat{w}_i^t + \varepsilon]$ liegt dar, d.h.

$$\hat{C} = \{\beta \in \mathcal{U} | f_t(\beta(v_1), ..., \beta(v_m)) = 0\} \cap \{\beta \in \mathcal{U} | \beta(v_i) = \hat{w}_i^t\}. \tag{4.4}$$

Auf diese Weise können natürlich auch mehrere Messungen simultan integriert werden. Unter Umständen müsste dann für jede Dimension ein separates ε gewählt werden. Zu beachten ist, dass die obigen Constraints C bzw. $\hat{C}$ von der Funktion f_t, der Variable v_i und gegebenenfalls von dem Parameter ε abhängen. Aus Gründen der Übersichtlichkeit verzichten wir aber im Weiteren auf das Anfügen der Indizes.

In manchen Fällen lässt sich ein Modell eines Sensors S_i direkt durch eine Funktion der Form

$$g_i^t : \Pi_{j=1,...,m,\ j \neq i} Dom(v_j) \rightarrow Dom(v_i) \subseteq \mathbb{R}. \tag{4.5}$$

angeben. Das bedeutet, dass die Funktion g_{S_i} die perfekten Messungen des Sensors S_i berechnet. Insbesondere gilt dann für alle Zusammenhänge f_t:

$$f_t(v_1, ..., v_{i-1}, g_i^t(v_1, ..., v_{i-1}, v_{i+1}, ..., v_m), v_{i+1}, ..., v_m) = 0 \tag{4.6}$$

Die Funktion g_i definiert offensichtlich selbst ebenfalls den Zusammenhang

$$g_i^t(v_1, ..., v_{i-1}, v_{i+1}, v_m) - v_i = 0. \tag{4.7}$$

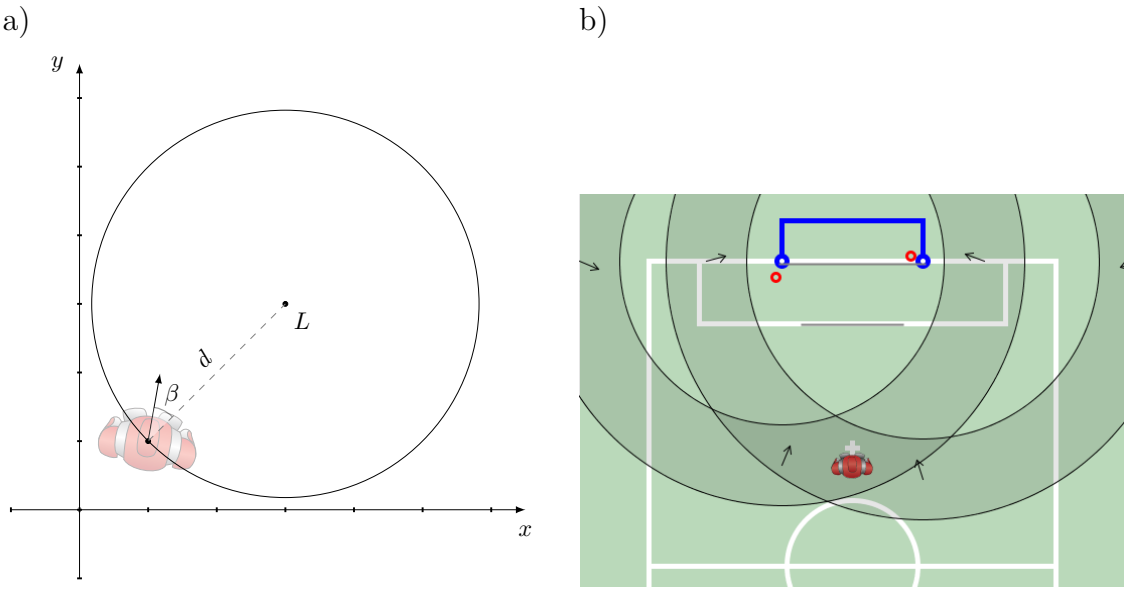


Abbildung 4.1.: *Der Roboter berechnet die Distanz und den horizontalen Winkel zu den gesehenen Torpfosten.*

Dann lässt sich die obige allgemeine Form eines Perzept-Constraints 4.3 wie folgt schreiben:

$$\hat{C} = \{\beta \in \mathcal{U} | g_i^t(\beta(v_1), ..., \beta(v_{i-1}), \beta(v_{i+1}), \beta(v_m)) - \beta(v_i) = 0 \wedge \beta(v_i) \in [\hat{w}_i^t - \varepsilon, \hat{w}_i^t + \varepsilon]\} \tag{4.8}$$

$$= \{\beta \in \mathcal{U} || g_i^t(\beta(v_1), ..., \beta(v_{i-1}), \beta(v_{i+1}), \beta(v_m)) - \beta(v_i)| \leq \varepsilon\} \tag{4.9}$$

Hier sollte man auch beachten, dass g_i^t abhängig von der Zeit t definiert ist. D.h. in diesem Sinne kann sich das Modell des Sensors mit der Zeit ändern.

In den folgenden Abschnitten untersuchen wir exemplarisch die Konstruktion von Constraints anhand einiger Sensordaten innerhalb des RoboCup. Wir beschränken uns dabei nur auf die Modellierung der Position des Roboters auf dem Spielfeld. Die Position des Roboters wird durch seine Koordinaten auf dem Spielfeld sowie seine Ausrichtung beschrieben. Die Variablen sind in diesem Fall $x \in [X_{min}, X_{max}]$, $y \in [Y_{min}, Y_{max}]$ und $\alpha \in [-\pi, \pi]$, wobei X_{min} die minimale und X_{max} die maximale mögliche Position des Roboters in x-Richtung ist (andere Grenzen sind analog zu interpretieren).

4.1.1. Pylon-Landmarken

Sei die Position einer Pylon-Landmarke[2] P durch die Koordinaten (x_P, y_P) gegeben, dann berechnen folgende Funktionen die Entfernung und den Winkel zu P

$$f_d^P(x, y, \alpha) := \|(x, y) - (x_P, y_P)\|_2 \tag{4.10}$$
$$f_\beta^P(x, y, \alpha) := \text{atan2}(y - y_P, x - x_P) - \alpha \tag{4.11}$$

Für eine gemessen Entfernung $\hat{d}$ und Winkel $\hat{\beta}$ zu der Landmarke P können wir also die entsprechende Constraints C_P^d und C_P^β wie folgt aufstellen

$$\hat{C}_P^d := \{(x, y, \alpha) \in \mathcal{U} || f_d^P(x, y, \alpha) - d| \leq \varepsilon_d\} \tag{4.12}$$
$$\hat{C}_P^\beta := \{(x, y, \alpha) \in \mathcal{U} || f_\beta^P(x, y, \alpha) - \beta| \leq \varepsilon_\beta\} \tag{4.13}$$

wobei ε_d und ε_β jeweils die zugelassenen Abweichungen sind. Die Abbildung 4.1 b) veranschaulicht die geometrischen Zusammenhänge, die für die Konstruktion der Constraints verwendet wurden.

Die Abbildung 4.1 zeigt eine typische Situation eines RoboCup Spiels in der *Standard Platform League*.

4.1.2. Linienlandmarken

Als eine *Linienlandmarke* bezeichnen wir eine Markierung auf dem Boden in Form eines Liniensegments, die der Roboter wahrnehmen kann. In RoboCup ist das Spielfeld mit weißen Linien ausgestattet. Diese Linien dienen als Motivation und Prototyp für solche Landmarken.

Wir beschrieben eine Linienlandmarke L durch einen Liniensegment, d.h. eine Gerade, die zwei Endpunkte besitzt. Ein Liniensegment wird durch den Vektor $(d, \beta, t_{min}, t_{max})$ definiert. Die Geradengleichung ist dann in Parameterform gegeben durch

$$g_L(t) := d \cdot \eta + t \cdot \nu \tag{4.14}$$

wobei

$$\eta = \eta(\beta) = \begin{pmatrix} -\sin\beta \\ \cos\beta \end{pmatrix}, \text{ und } \nu = \nu(\beta) = \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix} \tag{4.15}$$

Die Parameter t_{min} und t_{max} beschreiben dabei die beiden Enden des Segments. Das Liniensegment an sich ist dann also gegeben durch

$$L := \{g_L(t) | t \in [t_{min}, t_{max}]\} \subset \mathbb{R}^2 \tag{4.16}$$

Die Abbildung 4.2 veranschaulicht die Geometrie der Landmarke an einem Beispiel.

[2] Als *Pylon-Landmarken* bezeichnen wir die farbigen Markierungspfosten wie sie im RoboCup verwendet werden.

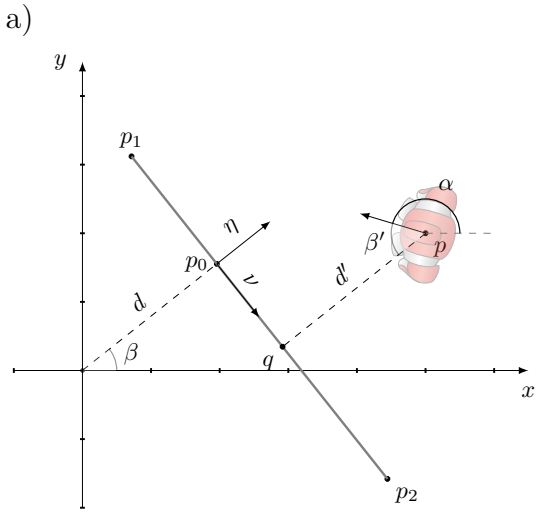


Abbildung 4.2.: *Die Geometrie einer Linienlandmarke wird illustriert. Die Gerade durch das Liniensegment wird mit g_L bezeichnet. Dann gilt für die Endpunkte $p_1 = g_L(t_{min})$, $p_2 = g_L(t_{max})$, $t_{min} = -\|p_1 - p_0\|$, $t_{max} = \|p_2 - p_0\|$. Außerdem gilt: die Position des Roboters $p = (p_x, p_y)$, Projektion auf die Gerade $q = g_L(\nu^T \cdot p)$, $p_0 = d \cdot \eta$, Winkel zu der Geraden $\beta' = g_t^{\beta}(p_x, p_y, \alpha)$, Distanz zwischen Roboter und der Geraden $d' = g_t^{d}(p_x, p_y, \alpha)$. Das Liniensegment in lokalen Roboterkoordinaten wird durch $(d', \beta', t'_{min}, t'_{max})$ beschrieben, wobei $t'_{min} = -\|p_1 - q\|$, $t'_{max} = \|p_2 - q\|$.*

Die Bildverarbeitung liefert nun die Information über ein gesehenes Liniensegment in der Form

$$(\hat{d}, \hat{\beta}, \hat{t}_{min}, \hat{t}_{max}) \in \mathbb{R} \times [-\pi, \pi] \times \mathbb{R}^2. \tag{4.17}$$

Diese Parameter beschreiben das gesehene Liniensegment in lokalen Koordinaten des Roboters. Die Abbildung 4.3 veranschaulicht die Geometrie des Linienperzepts.

Um das gesehene Linienperzept mit der globalen Landmarke vergleichen zu können, transformieren wir die Landmarke in die lokalen Koordinaten des Roboters. Dazu bestimmen wir zunächst die Entfernung des Roboters zur Linie

$$g^0(x, y, \alpha) := d_L - \eta_L^T \cdot (x, y)^T \tag{4.18}$$

Man beachte, dass diese Entfernung negativ ist, wenn der Roboter sich nicht in derselben Halbebene mit dem Koordinatenursprung befindet. Die Distanz und der Winkel zu der Landmarke lassen sich nun einfach berechnen durch

$$g_t^d(x, y, \alpha) := |g^0(x, y, \alpha)| \tag{4.19}$$

$$g_t^{\beta}(x, y, \alpha) := \frac{\pi}{2} + \alpha - \alpha_L \tag{4.20}$$

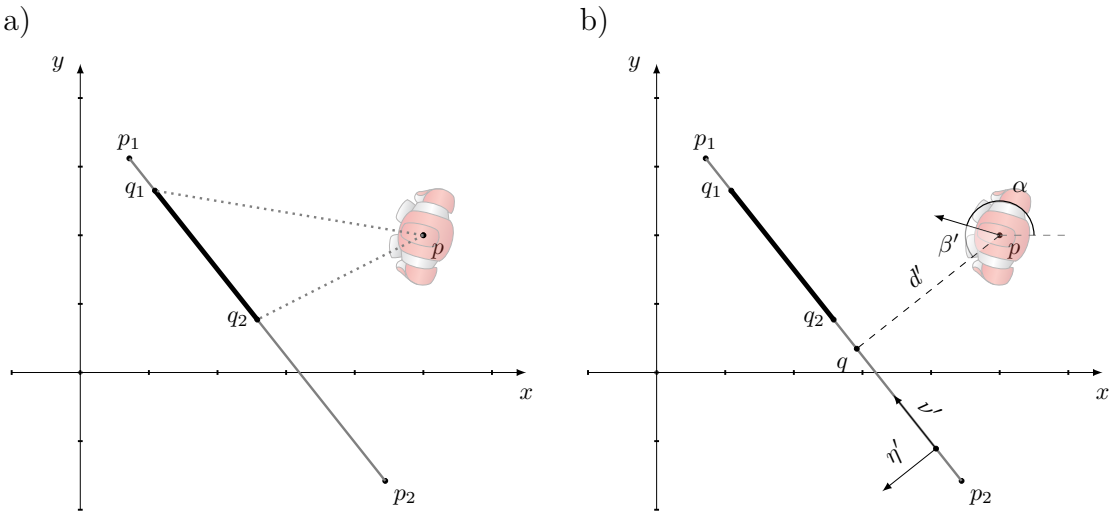


Abbildung 4.3.: *Eine Linienlandmarke aus der Sicht des Roboters. Die Abbildung links visualisiert den Liniensegment zwischen den Punkten q_1 und q_2 der von dem Roboter wahrgenommen wird. Der Segment wird aus der Sicht des Roboters beschrieben durch $(d', \beta', t'_{min}, t'_{max})$ wobei $t'_{min} = -\|q - q_2\|$ und $t'_{max} = \|q - q_1\|$, wie in der Abbildung rechts dargestellt.*

Um die neuen Parameter t'_{min} und t'_{max} der Endpunkte zu bestimmen, berechnen wir zunächst die Projektion q der Position des Roboters auf die Gerade. Es gilt:

$$g_L(\nu_L^T \cdot (x, y)^T) = q \tag{4.21}$$

Nun verschieben wir t_{min} und t_{max} gegenüber dem Nullpunkt. Gegebenenfalls müssen die Enden gespiegelt werden (weil der Roboter die Linie von der anderen Seite aus sieht)

$$g_t^{t_0}(x, y, \alpha) := \text{sign}(g_t^d(x, y, \alpha)) \cdot (t_{min}^L - \nu_L^T \cdot (x, y)^T) \tag{4.22}$$
$$g_t^{t_1}(x, y, \alpha) := \text{sign}(g_t^d(x, y, \alpha)) \cdot (t_{max}^L - \nu_L^T \cdot (x, y)^T) \tag{4.23}$$

Mit diesen Funktionen können wir nun die Landmarke in den lokalen Koordinaten des Roboters durch $(d', \beta', t'_{min}, t'_{max})$ angeben, wobei

$$d'(x, y, \alpha) := g_t^d(x, y, \alpha) \tag{4.24}$$
$$\beta'(x, y, \alpha) := g_t^\beta(x, y, \alpha) \tag{4.25}$$
$$t'_{min}(x, y, \alpha) := \min\{g_t^{t_0}(x, y, \alpha), g_t^{t_1}(x, y, \alpha)\} \tag{4.26}$$
$$t'_{max}(x, y, \alpha) := \max\{g_t^{t_0}(x, y, \alpha), g_t^{t_1}(x, y, \alpha)\} \tag{4.27}$$

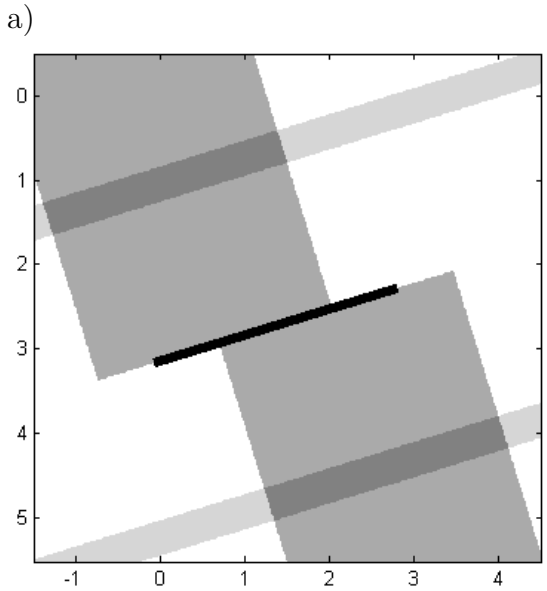


Abbildung 4.4.: *Die schwarze dicke Linie illustriert eine Linienlandmarke definiert durch* $(d, \beta, t_{min}, t_{max}) = (3, -0.3, -1, 2)$*. Wir nehmen an der Roboter hat folgende Werte gemessen* $(\hat{d}, \hat{\beta}, \hat{t}_{min}, \hat{t}_{max}) = (2, x, 0.5, 1)$*. In diesem Experiment wird nur die Einschränkung der Position betrachtet, jedoch nicht der Winkel. Die grauen Bereiche stellen die Einschränkungen dar, die sich aufgrund der gemessenen Distanz und der Verschiebung ergeben. Der hellgraue Bereich parallel zum Liniensegment ergibt sich aus der gemessenen Distanz* $\hat{d} = 2$ *und der Abweichung* $\varepsilon_d = 0.2$*. Der dunklere Bereich orthogonal zum Liniensegment folgt aus den Werten* $\hat{t}_{min} = 0.5, \hat{t}_{max} = 1$ *mit der Abweichung* $\varepsilon_t = 0.2$*. Der dunkelgraue Bereich stellt letztendlich den Durchschnitt beider Einschränkungen dar und somit die Einschränkung für die Position des Roboters.*

Damit lässt sich nun das Constraint wie folgt angeben

$$C_L := \{(x, y, \alpha) \in \mathcal{U} || g_t^d(x, y, \alpha) - \hat{d}| \leq \varepsilon_d \quad \wedge \tag{4.28}$$

$$|g_t^\beta(x, y, \alpha) - \hat{\beta}| \leq \varepsilon_\beta \quad \wedge \tag{4.29}$$

$$\hat{t}_{min}, \hat{t}_{max} \in [t'_{min}(x, y, \alpha) - \varepsilon_t, t'_{max}(x, y, \alpha) + \varepsilon_t]\} \tag{4.30}$$

Das Constraint C_L wird also insgesamt durch drei Bedingungen definiert: den Abstand zur Linie, die Verschiebung entlang der Linie und die Ausrichtung bezüglich der Linie. Die Abbildung 4.4 illustriert dieses Constraint an einem Beispiel. Aus Gründen der Übersichtlichkeit wird dabei die Bedingung für die Rotation des Roboters nicht dargestellt.

4.2. Algorithmen

In diesem Abschnitt beschreiben wir zwei konkrete Umsetzungen der BSCP für den Fall der Lokalisierung. Dabei werden die aus den Sensordaten gewonnenen Constraints über die Zeit integriert.

Wenn dem Roboter Information über seine Bewegungen (siehe *Odometrie* Abschnitt A.2) zur Verfügung steht, dann kann diese dazu verwendet werden um sein bisheriges Modell zu aktualisieren. Die neuen Constraints aus der Wahrnehmung (wie in dem Abschnitt 4.1 beschrieben) werden dann dazu verwendet das Modell zu korrigieren bzw. zu vervollständigen.

Damit lässt sich der Algorithmus klassischerweise in zwei Schritte unterteilen: Vorhersage (Integration von Bewegungsdaten) und Korrektur (Integration neuer Sensordaten). Im Folgenden werden beide Schritte genauer erläutert.

4.2.1. Vorhersage: Propagierung mit Odometrie

Wenn der Roboter sich bewegt werden die Constraints, die seine aktuelle Position modellieren, ebenfalls verschoben. Sei die Odometrie o nach der Definition aus Abschnitt A.2 gegeben. Dann können wir ein Constraint $C \in \mathcal{U}$ wie folgt verschieben:

$$C' = \{p \in \mathcal{U} | p = o \cdot q, q \in C\} \tag{4.31}$$

Für die Intervallconstraints lässt sich die Verschiebung einfach berechnen. So ergibt sich für ein Intervall $I = [a, b] \in \mathcal{I}$

$$I' = [o \cdot a, o \cdot b]. \tag{4.32}$$

Die Abbildung 4.5 illustriert die Verschiebung eines Intervallconstraints mit der Odometrie. Verschiebung von Intervallvereinigungen kann analog komponentenweise passieren.

Durch das Laufen, Rutschen auf dem Boden und Kollisionen wird die Odometrie verrauscht. Um diese Ungenauigkeiten abzubilden, werden die Constraints um einen entsprechenden Faktor $\lambda \geq 1$ vergrößert:

$$C' = \{p \in \mathcal{U} | p = \lambda \cdot q, q \in C\}. \tag{4.33}$$

Im Fall der Intervallconstraints können wir schreiben:

$$I' = I \cdot \lambda = [\lambda \cdot a, \lambda \cdot b]. \tag{4.34}$$

Der Vergrößerungsfaktor muss das Rauschen der Odometrie abbilden. Im einfachsten Fall kann er als eine Konstante experimentell bestimmt werden. Um genauere Resultate zu erhalten, könnte λ z.B. in Abhängigkeit von der aktuellen Geschwindigkeit oder Bewegungsart des Roboters bestimmt werden. Als weitere Verfeinerung

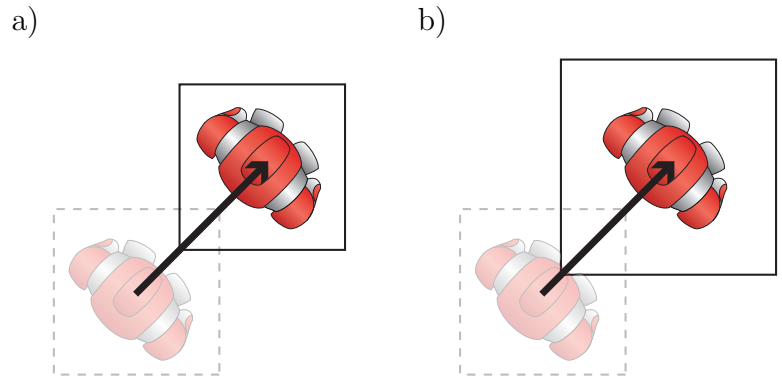


Abbildung 4.5.: *Einfache Propagierung eines Constraints mit Odometrie. Die Bewegung des Roboters ist durch den Pfeil dargestellt. Die gestrichelte Boxen stellen jeweils das initiale Constraint. a) Das Constraint wird mit den Odometrie-Daten verschoben. b) Die Bewegungsunsicherheit wird durch eine Vergrößerung des Constraints widergespiegelt.*

könnte der Vergrößerungsfaktor separat für jede Variable des Constraints gewählt werden. D.h. die Vergrößerung erfolgt abhängig von der Bewegungsrichtung.

4.2.2. Korrektur: Integration von Sensordaten

Verrauschte Sensordaten können zu Inkonsistenzen in den Constraints führen, d.h. es kann passieren, dass keine globale Lösung des Problems gefunden werden kann. Wie wir bereits im Abschnitt 3.2 diskutiert haben, gibt es unterschiedliche Möglichkeiten damit umzugehen.

Als erstes schauen wir uns an welche Constraints uns zur Verfügung stehen. Zunächst haben wir das mit der Odometrie vorhergesagte Constraint $\hat{C}_t^B$ zur Zeit t, das den aktuellen modellierten Zustand des Roboters repräsentiert. Zusätzlich haben wir die aus den aktuellen Sensordaten generierten Constraints $C_t^{z_1}, ..., C_t^{z_n}$, wobei $z_1, ..., z_n$ verschiedene Sensordaten bezeichnen.

Nun haben wir zu entscheiden, welche Constraints mit welchen propagiert werden sollen um möglichst gute Resultate zu erhalten. Im Folgenden stellen wir zwei mögliche Implementierungen des BSCP, bei denen wir unterschiedliche Strategien für die Integration der Constraints verfolgen, vor.

Direkte Integration (Greedy)

Eine einfache Möglichkeit der Propagierung ist die direkte Integration der Sensordaten in das aktuelle Modell. Dabei wird das aktuelle Modell iterativ mit den Sensor-Constraints propagiert solange das Resultat nicht leer ist:

Algorithmus 4 : Greedy propagation

Input : $\hat{C}_t^B, C_t^{z_1}, ..., C_t^{z_n}$
Result : C_t^B

$C_t^B \leftarrow \hat{C}_t^B$;
for $i = 1$ *to* n **do**
  $S \leftarrow C_t^B \cap C_t^{z_i}$;
  if $S \neq \emptyset$ **then**
    $C_t^B \leftarrow S$;
  end
end
return C_t^B

Der Vorteil dieses Ansatzes ist die Einfachheit. Wenn allerdings Constraints aus verrauschten Sensordaten generiert wurden, dann kann das zu sehr kleinen Schnittresultaten führen, sodass andere Constraints aufgrund der Inkonsistenz nicht mehr betrachtet werden. Insbesondere hängt das resultierende Constraint C_t^B von der Reihenfolge der Propagierung ab. Es kann also passieren, dass durch eine andere Reihenfolge bessere Resultate erhalten werden können.

Konsistente Integration

Um die Nachteile der direkten Integration, d.h. die Abhängigkeit von der Reihenfolge der Integration von Sensor-Constraints, zu eliminieren versuchen wir vor der Integration die beste Reihenfolge zu bestimmen. Dazu suchen wir zunächst eine maximale

konsistente Teilmenge der Sensor-Constraints und propagieren diese miteinander. Das Resultat wird dann mit der Vorhersage $\hat{C}_t^B$ propagiert:

Algorithmus 5 : Sensor Constraints Propagation

Input : $\hat{C}_t^B, C_t^{z_1}, ..., C_t^{z_n}$
Result : C_t^B

```
1  S ← C_t^{z_1} ;
2  for i = 2 to n do
3  |  S ← S ∩ C_t^{z_i};
4  end
5  if S ∩ Ĉ_t^B ≠ ∅ then
6  |  C_t^B ← Ĉ_t^B ∩ S ;
7  else
8  |  C_t^B ← increaseBoundaries(Ĉ_t^B)
9  end
10 return C_t^B
```

Wenn S und das Resultat $\hat{C}_t^B \cap S$ nicht leer sind, dann werden sie zum neuen Modell C_t^B zur Zeit t propagiert. Wenn aber S oder der Schnitt mit $\hat{C}_t^B$ leer ist, dann werden die Grenzen von $\hat{C}_t^B$ vergrößert und das Ergebnis als das neue Modell C_t^B ausgegeben. D.h. das aktuelle Modell wird durch die Sensordaten bestätigt und korrigiert. Falls aber die Sensordaten inkonsistent mit dem aktuellen Modell sind, wird nur die auf der Odometrie basierende Vorhersage $\hat{C}_t^B$, versehen mit zusätzlicher Unsicherheit, verwendet. Damit werden inkonsistente Sensordaten als „nicht korrekt“ ausgefiltert. Es kann aber passieren, dass die Sensordaten in der Realität korrekt sind, und die Inkonsistenzen dadurch verursacht werden, dass die Vorhersage $\hat{C}_t^B$ nicht der Realität entspricht, das ist zum Beispiel dann der Fall, wenn der Roboter ohne sein Wissen verschoben wurde (*kidnapped robot problem*). In diesem Fall werden alle Sensordaten solange abgelehnt, bis das Constraint C_t^B auf die Größe angewachsen ist, bei der es wieder mit Sensordaten konsistent ist. Dann werden die Sensordaten wieder integriert und das Model „springt“ an die richtige Stelle.

4.2.3. Behandlung von Inkonsistenzen

Der im letzten Abschnitt vorgestellte Algorithmus behandelt die inkonsistenten Daten als „falsche Messungen“, d.h. sie werden einfach ignoriert.

Die Vergrößerung von Constraints beim Auftreten inkonsistenter Sensordaten reicht nicht immer aus. Denn im Fall, dass der Roboter ohne sein Wissen auf eine andere Stelle versetzt wurde, kann es passieren, dass einige der Daten (z.B. aufgrund von Symmetrien) immer noch konsistent mit der alten Position sind. Dadurch wird das „Wachsen“ des Position-Constraints verhindert und der Roboter wechselt nie auf die neue Position.

Eine mögliche Lösung für dieses Problem ist es, die Inkonsistenzen explizit zu

beobachten. Genauer gesagt, können wir die konsistenten und die inkonsistenten Sensordaten zählen und daraus die Inkonsistenzrate der Daten[3]

$$IDR := \frac{\sharp inkonsistenteDaten}{\sharp konsistenteDaten} \tag{4.35}$$

berechnen. Wenn IDR kleiner als 1 ist, wird die Position nur mit konsistenten Daten aktualisiert (wie im Algorithmus 5). Wenn aber die Sensordaten inkonsistent werden, d.h. IDR wird groß (z.B. im Fall des *kidnapped robot problem*), dann nehmen wir die inkonsistenten Daten als mögliche Position des Roboters hinzu. Das ermöglicht dem Algorithmus auf die neue Position zu konvergieren.

In manchen Fällen wissen wir, dass einige Sensordaten verlässlicher sind als die anderen. Das kann z.B. an der Erkennung liegen oder auch daran, dass einige Daten kleinere Constraints produzieren als die anderen. So ist die Erkennung der Torpfosten im RoboCup viel zuverlässiger als die Erkennung von Linien. Das hängt damit zusammen, dass die Tore eine unverwechselbare Form und Farbe besitzen während es vorkommen kann, dass Linien falsch erkannt werden. Andererseits liefern die Linien relativ mehrdeutige Constraints (vgl. Abbildung 4.6), was hauptsächlich an der Symmetrie der Feldlinien liegt. Dieses Wissen kann als Heuristik für die Auswahl von Sensordaten bei der Propagierung verwendet werden.

4.2.4. Schätzung der Position des Roboters

Das Ergebnis der Propagierung von Constraints ist ein Constraint (also eine Einschränkung), das die möglichen Positionen des Roboters beschreibt, die zu den gemachten Beobachtungen passen. In unserer Umsetzung erhalten wir also eine Vereinigung von Intervallen als Beschreibung der aktuellen Position des Roboters. Wenn die Sensorinformation es nicht erlaubt die Position genau einzuschränken, kann dieses Constraint sehr groß werden und aus mehreren disjunkten Komponenten bestehen.

Die Art der Positions-Information die benötigt wird hängt stark von der Anwendung ab. Bei einigen Fragestellungen wie z.B. *„Bin ich in der eigenen Hälfte?“* könnte das Positions-Constraint direkt verwendet werden. Es gibt aber auch Anwendungen, bei denen es erforderlich ist, eine konkrete Schätzung der Position des Roboters zu haben. Dann muss aus dem Lösungsconstraint eine Lösung ausgewählt werden. Wenn das Constraint aus einem Intervall besteht, kann das Zentrum des Intervalls als Schätzung für die Position benutzt werden. Wenn es mehrere Komponenten gibt, dann kann das Zentrum der größten Komponente genommen werden. Diese einfache Methode haben wir verwendet um die Constraint-Lokalisierung mit MCPF vergleichen zu können (Ergebnisse werden im nächsten Abschnitt vorgestellt).

[3]Inconsistent Data Ratio

4.3. Implementierung und Experimente

Die Implementierung wurde in zwei Schritten durchgeführt. Als erstes wurden die vorgestellten Algorithmen in einem speziell dafür entwickelten abstrakten Simulator implementiert und getestet um die grundsätzliche Umsetzbarkeit zu überprüfen. Im zweiten Schritt wurde eine Lokalisierung für einen realen Roboter implementiert und getestet. Als Plattformen dienten dabei die Roboter Aibo ERS-7 und Nao V3+.

Die Ziele der Experimente waren dabei neben grundsätzlicher Umsetzbarkeit die Laufzeit des Algorithmus und die Genauigkeit der Lokalisierung.

In folgenden Experimenten im RoboCup Umfeld vergleichen wir eine Implementation des Monte Carlo Partikelfilters (MCPF) mit der Constraint-basierter Lokalisierung (wie oben beschrieben). Besondere Aufmerksamkeit wird dabei auf die Laufzeit und Lokalisierungsgenauigkeit gerichtet.

4.3.1. Simulation

Für die Entwicklung und der Test der Constraint-basierten Lokalisierung wurde ein abstrakter Simulator entwickelt. Das erlaubt uns die Verfahren schrittweise zu simulieren und in allen Details zu beobachten. Während eines Experiments können dabei folgende Elemente simuliert werden:

- Objekte: Linien, Pylon-Landmarken, Roboter, etc. (beliebig erweiterbar).
- Bewegung des Roboters in der Welt
- Wahrnehmung des Roboters: Abstand und Winkel zu den Linien und Torpfosten, etc.
- eingeschränkte Sicht
- Rauschen der Sensoren

Dabei wird keine physikalische Simulation durchgeführt. Zusätzlich gibt es eine Reihe von Werkzeugen, die die Durchführung und die Überwachung der Experimente erleichtern. Hier ist ein kurzer Überblick über einige wichtige Funktionen des Simulators:

- Infrastruktur: der Simulator bietet eine Infrastruktur, die es erlaubt Experimente je nach Bedarf zu erstellen. Man kann dabei verschiedene Welten, Sensoren und eingesetzte Verfahren kombinieren. Abbildung 4.7 veranschaulicht zwei verschiedene Experimente;
- Logplayer: erlaubt das Aufnehmen und wieder Abspielen einer Bewegungstrajektorie des Roboters. Dabei werden alle Sensordaten aufgezeichnet, sodass Reproduzierbarkeit des Experiments gewährleistet ist;

a) b)

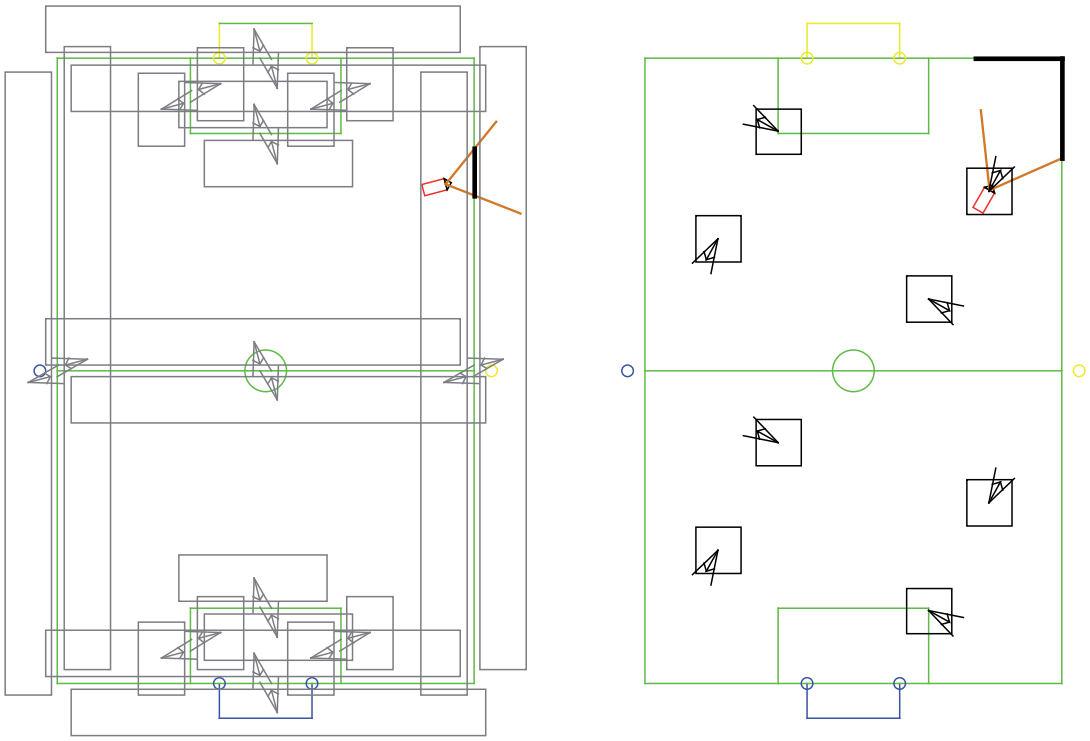


Abbildung 4.6.: *Roboter positioniert auf einem Fußballfeld. Dicke schwarze Linien markieren die vom Roboter gesehenen Liniensegmente. a) Graue Boxen stellen die Constraints dar, die aus einem einzigen Liniensegmenten generierte wurden. b) Zwei Constraints werden aus gesehenen Liniensegmenten generiert (nicht im Bild). Die schwarzen Boxen stellen das Ergebnis der Propagierung beider Constraints dar.*

- Plotten: es gibt die Möglichkeiten für eindimensionale Plots (z.B. für die Darstellung des Lokalisierungsfehlers) und Trace Plots, die das Plotten der Spur des Roboters auf dem Feld erlaubt;

- Visualisierung: durch ein generisches Konzept der *Drawings* ist es möglich beliebige Visualisierungen zu realisieren. Dazu wird an einer beliebigen Stelle im Algorithmus ein Drawing erstellt und in einen globalen Pool hinzugefügt. Der Simulator sorgt dann dafür, dass die Zeichenroutine des Drawings zum richtigen Zeitpunkt aufgerufen wird. Alle Drawings können mit einer CheckBox ein- und ausgeblendet werden. Ein Drawing kann dabei je nach Bedarf implementiert werden, so ist die Visualisierung der Boxen in der Abbildung 4.6 ein Beispiel für ein solches Drawing;

a) b)

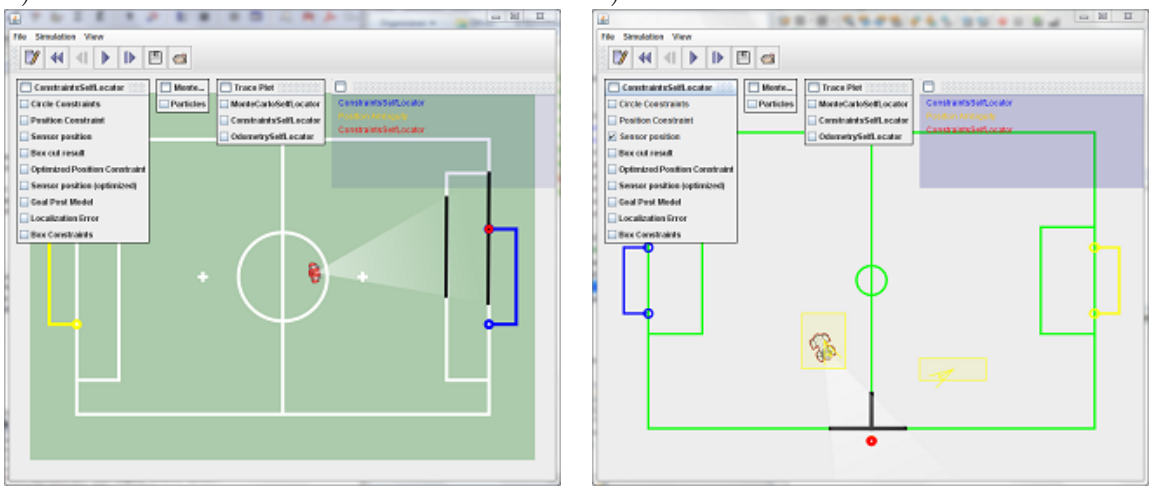


Abbildung 4.7.: *Zwei Beispielexperimente im Simulator. Es ist das Bedienpanel des Logplayers sowie verschiedene Menüs zum anzeigen von Debugdrawings zu sehen. (links) ein Nao Roboter sieht Linien und einen Torpfosten. (rechts) ein Aibo Roboter sieht die Linien und die Landmarke. Die Positions-Constraints die aus diesen Daten resultieren werden als Boxen visualisiert.*

Die Abbildung 4.6 a) veranschaulicht Constraints die aus gesehenen Liniensegmenten generiert wurden. In diesem Beispiel werden keine Fehler in den Sensordaten simuliert, sodass die Messungen absolut genau sind.

4.3.2. Vergleich mit der Monte-Carlo Lokalisierung

Als erstes sollte man anmerken, dass der allgemeine Vergleich der Laufzeit beider Algorithmen nicht ganz einfach ist, denn die Geschwindigkeit hängt stark von der Implementierung ab.

Bei diesem Experiment bewegte sich der Roboter auf einem vorgegebenen Pfad. Die Position des Roboters wurde dabei mit MCPF und dem Constraint basierten Ansatz modelliert. In jedem Schritt haben wir die jeweils modellierte Position mit der exakten verglichen und die benötigte Rechenzeit beider Algorithmen protokolliert.

Wie man in der Abbildung 4.8 gut sehen kann zeigte die Zeitmessung, dass der Constraint basierte Ansatz (MCI) etwa 5-10 mal schneller war als der Partikelfilter. Außerdem kann man sehen, dass die Berechnungszeit des Partikelfilters stärker variiert als die des Constraint Algorithmus.

Die Abbildung 4.9 veranschaulicht die Spuren der jeweils modellierten Positionen im Vergleich mit der exakten Trajektorie des Roboters. Insgesamt ist die Abweichung in beiden Fällen vergleichbar. An einigen Stellen scheint der Constraint basierte Algorithmus empfindlicher auf die Fehler in den Sensordaten zu reagieren, was sich

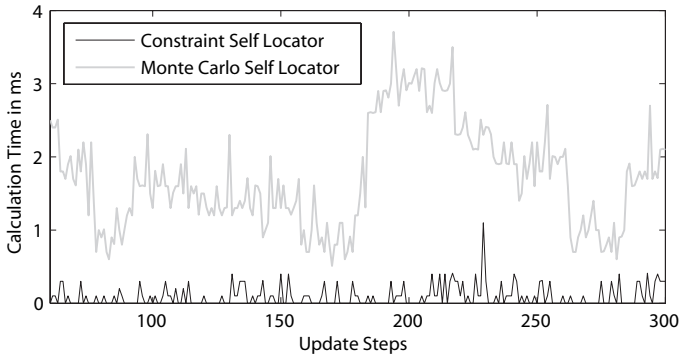


Abbildung 4.8.: *Berechnungszeit für einen Modellierungsschritt auf einem 1.5 GHz Prozessor. Graue Linie: Monte Carlo Partikelfilter mit 100 Partikeln. Schwarze Linie: Constraint basierte Lokalisierung.*

in den Sprüngen der modellierten Position widerspiegelt wie in der Abbildung 4.9 b) gut zu sehen ist.

4.3.3. Experimente auf der Aibo-Plattform

Als erste Plattform für unsere Experimente diente der Roboter Aibo ERS-7 (Abschnitt B.1). Die Detektion der Linien ist durch die niedrige Höhe des Roboters und die geringe Auflösung der Kamera stark limitiert, sodass relativ wenige Sensordaten für die Lokalisierung verfügbar waren.

Die Abbildung 4.10 veranschaulicht eine Situation, in der dem Roboter eine Pylon-Landmarke und eine Linie als Information zur Verfügung stehen. In diesem Fall liefert der Constraint basierte Ansatz eine bessere Repräsentation aller möglichen Positionen als MCPF mit 100 Partikeln.

Solche Situationen sind für die Partikelfilter sehr schwierig, denn es bedarf einer großen Anzahl von Partikeln um große Verteilungen darzustellen. Insbesondere wenn es mehrere Alternativen für die Position des Roboters gibt (wie in der Abbildung 4.10), neigen die Partikelfilter mit wenigen Partikeln dazu eine Position zu bevorzugen.

Die Laufzeit beider Verfahren war vergleichbar.

4.3.4. Experimente auf der Nao-Plattform

Eine weitere Experimentalplatform stellt der humanoide Roboter Nao V3+ dar (Abschnitt B.2).

In diesem Experiment untersuchen wir den Einfluss verschiedener Constraints auf die Qualität der Lokalisierung. Der Roboter ist dabei seitlich vor einem Tor positioniert und bewegt seinen Kopf um die Umgebung zu erfassen. Als Sensorinformation stehen ihm dabei die Torpfosten und die Linien des Strafraums die durch die Bild-

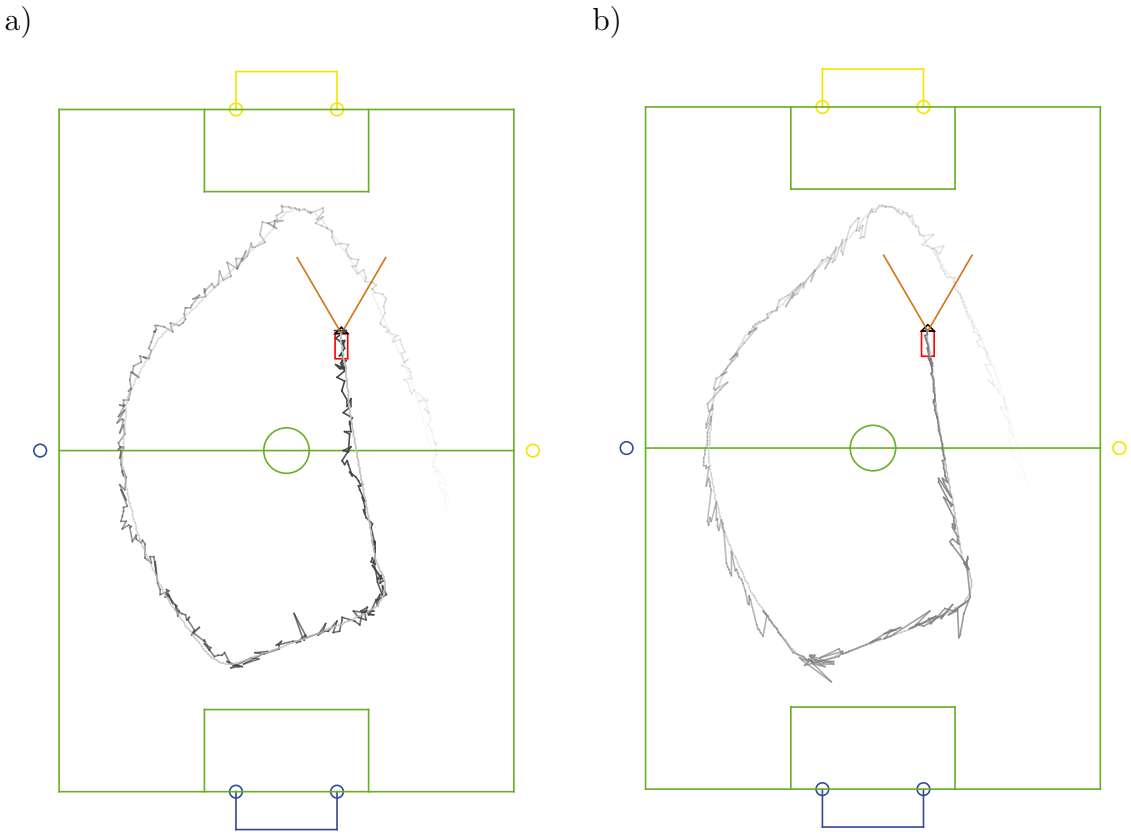


Abbildung 4.9.: *Vergleich der Lokalisierungsgenauigkeit in der Simulation: Ein Roboter bewegt sich auf dem Feld im Kreis. Die genaue Spur der Bewegung des Roboters ist durch die glatte graue Linie markiert. a) Die Spur der Monte-Carlo Partikelfilter basierten Lokalisierung. b) Constraint basierte Lokalisierung.*

verarbeitung detektiert werden zur Verfügung. Der Roboter kann jedoch nicht die gesamte Szene gleichzeitig erfassen, so kommen die Daten nacheinander rein.

Eine Besonderheit bei diesem Experiment ist, dass an einigen Stellen von dem Roboter Linien erkannt wurden, die so nicht vorhanden sind. Das führt dazu, dass Constraints aufgebaut werden die inkonsistent mit der realen Position des Roboters sind.

Als Maß für die Qualität der berechneten Position wurde die Fläche des Positions-Constraints verwendet.

Der Verlauf des Experiments ist in der Abbildung 4.11 zusammengefasst. Die obere Zeile zeigt die Veränderung der Fläche der berechneten Position abhängig von den Sensordaten. In den unteren Zeilen sind die Perzepte notiert, die zu dem jeweiligen Zeitpunkt wahrgenommen wurden.

Zusätzlich wurde die Anzahl konsistenter und inkonsistenter Perzept-Constraints

a) b)

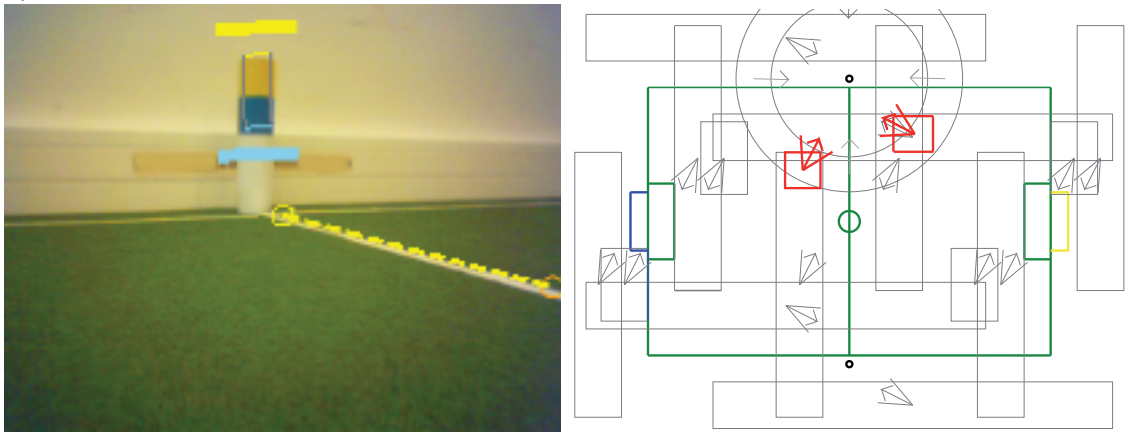


Abbildung 4.10.: *Experiment auf einem Aibo ERS-7: a) Szene aufgenommen von der Kamera des Roboters. In dem Bild wird eine Landmarke und eine Linie detektiert (wie im Bild markiert). b) Die Information aus dem Bild a) wird dazu benutzt zwei Constraints für die Position des Roboters aufzubauen (graue Boxen entlang der Linien und der Ring um die Landmarke). Nach der Propagierung beider Constraints ergeben sich zwei mögliche Positionen des Roboters (markiert durch dicke Rechtecke).*

(bezüglich der modellierten Position) notiert. Man kann sehr gut beobachten wie die Fläche wächst, wenn keine konsistenten Perzepte vorhanden sind. Die markierten Bereiche illustrieren diese Zeitabschnitte.

Zusammenfassend kann man sagen, dass das Vorhandensein falscher (inkonsistenter) Sensordaten nicht gravierend ist solange genügend konsistente Daten vorhanden sind. Das entspricht auch der Natur des Algorithmus 5.

4.4. Abschließende Bemerkungen

Die Leistung einer Implementierung eines Algorithmus hängt meistens sehr stark von der art der Umsetzung ab. Insbesondere haben die Parameter und die Effizienz der Umsetzung einen großen Einfluss auf die Genauigkeit und die Geschwindigkeit. Das macht es schwer allgemeine Aussagen zu treffen, oder direkte Vergleiche mit anderen Algorithmen zu machen.

Wir haben gezeigt, dass räumliche Constraints automatisch aus den Sensordaten generiert werden können. Als Beispiel haben wir Pylon-Landmarken und Linien dazu verwendet Constraints für die Position des Roboters aufzubauen. Wir haben konkrete Umsetzungen des Propagierungsschemas BCPS für den Spezialfall der Lokalisierung vorgestellt und Implementiert. Besonderes Augenmerk gilt dabei der Behandlung der Inkonsistenzen, die Aufgrund von verrauschten Daten entstehen. In

Abbildung 4.11.: *Von oben nach unten: (1) die Fläche des berechneten Positionsconstraints als Maß für die Qualität/Mehrdeutigkeit; (2),(3) zeigt an, ob jeweils der linke oder der rechte Pfosten des Tores gesehen wurde; (4) gibt die Anzahl der gesehenen Linien an; (5),(6) stellt die Anzahl der Perzept-Constraints, die mit dem Positions-Constraint konsistent bzw. inkonsistent sind dar. Graue Boxen markieren die Zeitabschnitte in denen falsche Tor-Perzepte detektiert wurden. In diesen Fällen sind alle gesehenen Perzepte inkonsistent und die Fläche des Positions-Constraints wächst bis korrekte Sensordaten detektiert werden.*

unserer Implementierung wurden Constraints in Form von Ringen und Mengen von Intervallen verwendet.

In den Experimenten haben wir gesehen, dass Constraint basierte Modellierung einen vielversprechenden Ansatz für die Modellierung der Umwelt darstellt. In der Simulation bietet das Constraint basierte Verfahren bereits jetzt eine Alternative zum MCPF. Sowohl unter dem Gesichtspunkt der Berechnungskomplexität als auch hinsichtlich der Genauigkeit der Lokalisierung. In den Fällen, wenn es darum geht mehrdeutige Position des Roboters zu repräsentieren, liefern die Constraints bessere Resultate als MCPF. In speziellen Situationen liefert der Constraint Ansatz auch auf den realen Robotern zu MCPF vergleichbare Resultate. Allerdings hat sich unsere Implementierung insgesamt als noch relativ empfindlich gegenüber von Fehlern in der Wahrnehmung gezeigt. Die resultierenden Inkonsistenzen stellen nach wie vor ein großes Problem des Ansatzes dar. Es hat sich auch gezeigt, dass die Bildung der Intervallhülle unabkömmlich ist. Anderenfalls wächst die Anzahl der Constraints bei Propagieren quadratisch an, was dazu führt das das Verfahren bereits nach sehr kurzer Zeit eine Speicherüberlauf produziert.

In unserer Implementierung haben wir nur einen Teil der Information verwendet die uns zur Verfügung stand. Viele Sensorinformationen und Zusammenhänge (wie kinematische Kette) können noch benutzt werden um die Genauigkeit und Stabilität zu verbessern. Eine weitere Einschränkung der vorgestellten Umsetzung stellt der Zustandsraum dar. Es wurde nur die Position des Roboters modelliert, wodurch viele Freiheitsgrade bei der Suche nach der Lösung verloren gehen.

Die vermutete Stärke des Ansatzes liegt in der Modellierung vieldimensionaler Zustände unter Verwendung vieler Zusammenhänge. Es bedarf noch weiterer Untersuchungen, um festzustellen welche Auswirkungen ein größerer Zustandsraum und eine größere Anzahl von Constraints auf die Leistung des Ansatzes haben kann.

5. Zusammenfassung und Ausblick

Am Anfang dieser Arbeit haben die Vorstellung von der Modellierung der Umwelt eines mobilen Roboers auf eine sehr allgemeine Basis gestellt. Wir betrachten diese Aufgabe als ein Optimierungsproblem, in dem bekanntes Wissen in Form von Einschränkungen auftritt. Die gesuchten Größen treten als Variablen des Problems auf. Die von den Sensoren gemessenen Größen können ebenfalls als Variablen des Problems optimiert werden. Die Messungen liefern weitere Einschränkungen für diese Variablen. Diese Sicht gibt uns den nötigen formalen Freiraum für unsere Ansätze.

Die Zentrale Frage der Arbeit ist:

> „Wie können (insbesondere redundante) Informationen besser ausgenutzt werden?“

Um diese Frage zu beantworten geht die Arbeit in zwei Richtungen. Im ersten Teil wurden redundante Information (wie die Ballgröße) dazu benutzt fehlerhafte Sensorwerte (wie etwa der Neigungswinkel der Kamera) zu korrigieren. Im zweiten Teil wurde ein auf Constraints basierender Ansatz für die Modellierung der Umwelt entwickelt.Wir fassen die Arbeit etwas detaillierter zusammen:

Im ersten Abschnitt wurden einige Möglichkeiten analysiert, wie die Eigenschaften der Objekte in der Umgebung des Roboters dazu genutzt werden können die Wahrnehmung zu verbessern. Insbesondere wurden Objekte mit bekannten Eigenschaften als Referenz genutzt um andere Objekten zu lokalisieren. Die Zusammenhänge zwischen dem Roboter und den Objekten wird dabei analytisch beschrieben. Wenn genügend viele Größen gegeben sind, können andere analytisch bestimmt werden. Wir haben gesehen, dass es viele Zusammenhänge gibt, die effektiv für die Lokalisierung genutzt werden können.

Im zweiten Abschnitt wurden theoretische Grundlagen für Constraint basierte Modellierung gelegt. Es wurden Kriterien für die Optimalität von Constraint-Netzen formuliert. Wir haben diskutiert wie die Constraints propagiert werden können. Insbesondere wurden dabei die Intervall-Constraints näher untersucht.

Im letzten Abschnitt haben wir den Constraint basierten Ansatz für die Aufgabe der Lokalisierung eines Roboters umgesetzt. Wir haben gesehen, dass Sensorinformation verwendet werden kann um Constraints aufzubauen. Es wurden Propagierungs-Algorithmen vorgeschlagen und Möglichkeiten diskutiert mit Inkonsistenzen umzugehen. In den Experimenten wurde gezeigt, dass die Constraints einen vielversprechenden Ansatz für die Modellierung der Umwelt darstellen. Die Genauigkeit und die Laufzeit wurde in der Simulation und auf realen Roboter Plattformen getestet und mit dem klassischen Monte-Carlo Partikelfilter verglichen.

Wir haben gesehen, dass durch gezielte Ausnutzung objektspezifischer Eigenschaften die Wahrnehmung insgesamt verbessert werden kann. Wir haben auch gesehen, dass Constraints für die Modellierung der Welt eingesetzt werden können.

5.1. Weitere Arbeit

Dass das von dieser Arbeit umfasste Gebiet ist offensichtlich viel zu groß, um in diesem Rahmen vollständig behandelt werden zu können. Viele Aspekte benötigen noch tiefergehender und umfassenderer Untersuchungen. In der Menge aller Möglichkeiten lassen sich folgende konkrete Richtungen besonders unterstreichen:

- Die vorgestellte Implementierung des Constraint-Ansatzes beschrankt sich auf die Modellierung der Position der Roboters. Damit konnten die erwarteten Vorteile der Constraints nicht vollständig entfaltet werden (Gleichzeitiges Modellieren vieldimensionaler Zustände). Eine zentrale Richtung für weitere Forschung ist daher die Ausweitung der Constraints auf mehr Variablen, z.B. simultane Modellierung der Position des Roboters und des Balls. Insbesondere aber auch Integration der im ersten Teil vorgestellten Zusammenhänge in den Constraint Ansatz (z.B. der Neigungswinkel wird als eine der Variablen modelliert).

- Für einige Anwendungen ist es wichtig eine eindeutige Lösung des Problems zu haben. Daher ist es von Interesse die Position des Roboters innerhalb des Constraints, der seine Position modelliert, so gut wie möglich zu bestimmen.

- An vielen Stellen hängen die umgesetzten Algorithmen entscheidend von bestimmten Parametern ab. Beispiele sind die Abweichungen ε bei der Konstruktion von Constraints oder der Vergrößerungsfaktor δ bei der Bildung der Intervall-Hülle. Eine optimale Wahl der Parameter könnte die Leistung der Lokalisierung deutlich verbessern. Hier könnten z.B. evolutionäre Verfahren in der Simulation eingesetzt werden um die Parameter zu schätzen.

- Es sind weitere Untersuchungen bezüglich der Behandlung der Inkonsistenzen erforderlich.

A. Einige Grundlagen

In diesem Abschnitt werden einige handwerkliche Grundlagen beschrieben, die in der Arbeit Verwendung finden.

A.1. Lösung von Gleichungen

In vielen Fällen lassen sich Gleichungen aufstellen, die die geometrischen Zusammenhänge zwischen den Parametern der Kamera (z.B. Neigungswinkel der Kamera, Rotationswinkel der Kamera an der optischen Achse etc.) und den Beobachtungen im Bild (z.B. Position und Größe des Balls im Bild) beschreiben. Dann lässt sich das Problem als ein Nullstellen-Problem auffassen, d.h. der gesuchte Parameter kann als eine Nullstelle einer Gleichung ermittelt werden. Die meisten Fragestellungen mit denen wir uns beschäftigen, führen zu Funktionen die unendlich oft differenzierbar und periodisch sind.

Es existiert eine Reihe von Methoden um Lösungen solcher Gleichungen zu bestimmen. Die meisten heute verwendeten Methoden zur numerischen Bestimmung von Nullstellen basieren auf dem *Newton Verfahren*. Bei dem Newton Verfahren wird mit folgender Vorschrift eine Lösung iterativ angenähert

$$\begin{cases} x_{j+1} := x_j + z_j \\ \text{mit } z_j \text{ aus } f'(x_j) \cdot z_j = -f(x_j) \end{cases} \tag{A.1}$$

für $j \geq 0$. Der Startwert x_0 muss dabei gewählt werden. Das Verfahren konvergiert quadratisch[1], wenn der Startwert nah genug an der Nullstelle gewählt ist. Mit dem Verfahren kann nur eine Nullstelle bestimmt werden. Wenn die Funktion f mehrere Nullstellen besitzt, dann entscheidet die Wahl des Startwertes x_0 im wesentlichen darüber welche Nullstelle gefunden wird.

Die Wahl des Startwertes ist ein entscheidender Faktor für die Geschwindigkeit des Verfahrens und dafür welche Nullstelle ermittelt wird, falls die Funktion mehrere Nullstellen besitzt.

Meistens unterliegt der gesuchte Wert bestimmten Einschränkungen, so hat z.B. ein Gelenk nur einen begrenzten Bereich in dem es sich bewegen kann. Diese Einschränkungen können helfen die Mehrdeutigkeit der Lösungen aufzulösen.

[1] Quadratische Konvergenz liegt vor, wenn für alle j die Abschätzung $|x_{j+1} - x_*| \leq C|x_j - x_*|^2$ mit einer Konstanten C gilt. Dabei bezeichnet x_* die Nullstelle.

In den meisten Fällen muss die Gleichung in einem Takt immer wieder gelöst werden (z.B. jedes mal wenn ein neues Bild von der Kamera kommt). Dann kann als Startwert der im letzten Schritt ermittelte Wert verwendet werden. Bei hohen Taktraten (z.B. beim Aibo 30 Bilder pro Sekunde) ändert sich der gesuchte Wert in der Regel nur wenig, somit liegt der Startwert sehr nah bei der Nullstelle, was dazu führt, dass die Lösung bereits nach wenigen Iterationen mit ausreichender Genauigkeit ermittelt wird. Zusätzlich löst solche Wahl des Startwertes teilweise das Problem der Mehrdeutigkeit möglicher Lösungen. Denn die Scheinlösungen liegen meistens relativ weit weg von der echten Lösung. Wird als ein mal die richtige Lösung gefunden, so liegt der Startwert in allen nachfolgenden Schritten in der Nähe der richtigen Lösung.

Sehr oft gibt es noch andere (meistens weniger genaue) Möglichkeiten den gesuchten Wert zu ermitteln, z.B. können die Gelenkwinkel oft durch das Berechnen der kinematischen Kette ermittelt werden. Diese Werte können ebenfalls als Startwerte für das Newton Verfahren verwendet werden um bessere Konvergenz zu erzielen und um die eventuelle Mehrdeutigkeiten aufzulösen.

A.2. Odometrie

Unter *Odometrie*[2]-Daten verstehen wir die Information über den von dem Roboter zurückgelegten Weg. Es gibt verschiedene Wege diese Information zu gewinnen, z.B. mit Hilfe verschiedener Sensoren wie Beschleunigungsvektoren, Radencoder, Drucksensoren an den Füßen oder Gyrometer. Die kinematische Kette und die Bewegungs-Kommandos können ebenfalls dazu verwendet werden, die Odometrie-Daten zu bestimmen. In den meisten Fällen wird je nach Verfügbarkeit und Genauigkeit eine Kombination dieser Daten benutzt.

Wir nehmen an, dass der Roboter in der Lage ist sich omnidirektional zu bewegen. D.h. wir können die Translation und Rotation des Roboters unabhängig voneinander modellieren. Damit lässt sich die Position des Roboters auf dem Feld durch eine Koordinatentransformation beschreiben. Sei

$$p = (x, y, \alpha) \in \mathbb{R}^2 \times [-\pi, \pi] \tag{A.2}$$

eine Position des Roboters, dann können wir p äquivalent als Matrix darstellen

$$\hat{p} = \begin{pmatrix} R(\alpha_p) & t_p \\ 0 & 1 \end{pmatrix} \tag{A.3}$$

wobei $R(\alpha_p) \in S_2(\mathbb{R})$ die Rotationsmatrix um den Winkel α_p ist und $t_p = (x, y)^t$ der Translationsanteil von p. Zur Vereinfachung verwenden wir die Schreibweisen A.2 und A.3 äquivalent.

[2]Wegmessung vom Griechischen *hodós*, „Weg“ und *métron*, „Maß“

Seien also $p, q \in U$ zwei Positionen. Wir suchen nun eine Transformation o so, dass

$$o \cdot p = q \tag{A.4}$$

$$\Leftrightarrow \begin{pmatrix} R(\alpha_o) & t_o \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} R(\alpha_p) & t_p \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R(\alpha_q) & t_q \\ 0 & 1 \end{pmatrix} \tag{A.5}$$

$$\begin{aligned} \Rightarrow \begin{pmatrix} R(\alpha_o) & t_o \\ 0 & 1 \end{pmatrix} &= \begin{pmatrix} R(\alpha_q) & t_q \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} R(\alpha_p) & t_p \\ 0 & 1 \end{pmatrix}^{-1} \\ &= \begin{pmatrix} R(\alpha_q) & t_q \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} R(-\alpha_p) & -R(-\alpha_p) \cdot t_p \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R(\alpha_q) \cdot R(-\alpha_p) & -R(\alpha_q) \cdot R(-\alpha_p) \cdot t_p + t_q \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} R(\alpha_q - \alpha_p) & t_q - R(\alpha_q - \alpha_p) \cdot t_p \\ 0 & 1 \end{pmatrix} \end{aligned}$$

Wenn also p eine vergangene und q die aktuelle Position des Roboters darstellt, dann beschreibt o die Odometrie des Roboters für den Übergang von p nach q.

A.3. Arcus Tangens: atan2

Die Funktion

$$\text{atan2} : \mathbb{R}^2 \longrightarrow [-\frac{\pi}{2}, \frac{\pi}{2}] \tag{A.6}$$

ordnet jedem Punkt $(x, y) \in \mathbb{R}^2$ den Winkel zwischen der positiven x-Achse und dem Vektor (x, y) zu. Wir definieren atan2 durch

$$\text{atan2}(y, x) := \begin{cases} \arctan \frac{y}{x} & \text{für } x > 0 \\ \arctan \frac{y}{x} + \pi & \text{für } x < 0,\ y \geq 0 \\ \arctan \frac{y}{x} - \pi & \text{für } x < 0,\ y < 0 \\ +\pi/2 & \text{für } x = 0,\ y > 0 \\ -\pi/2 & \text{für } x = 0,\ y < 0 \\ 0 & \text{für } x = 0,\ y = 0 \end{cases} \tag{A.7}$$

Die Motivation für die Verwendung von atan2 statt der klassischen Funktion arctan liegt darin, dass arctan nicht zwischen gespiegelten Vektoren unterscheiden kann, z.B. wird für die Vektoren $(1, 1)$ und $(-1, -1)$ berechnet

$$\arctan(1/1) = \arctan(-1/-1) = \frac{\pi}{4} \tag{A.8}$$

Ein zusätzliches Problem für arctan stellen die Vektoren der Form $(0, x)$ dar, denn in diesem Fall müsste durch 0 dividiert werden.

B. Roboter

In diesem Abschnitt stellen wir die drei Roboter vor, die als Plattform für unsere Experimente gedient haben. Alle Roboter haben folgende wesentliche Gemeinsamkeiten, die für unsere Untersuchungen wichtig sind:

1. bewegen sich auf Beinen (Laufroboter)
2. haben einen beweglichen Kopf
3. sind mit einer eingeschränkten gerichteten Kamera ausgestattet
4. sind dafür konzipiert autonom zu agieren

B.1. Aibo ERS-7

Der *Aibo ERS-7* (Abbildung B.1) ist die dritte Generation der *Aibo* Roboter, hergestellt von der Firma *Sony*. Dieses Modell wurde von 2003 bis 2006 produziert und diente von 2004 bis 2008 als Plattform in der *Standard Platform League* des RoboCup. Ab 2008 wurde *Aibo ERS-7* durch den Humanioden Roboter *Nao* B.2 ersetzt.

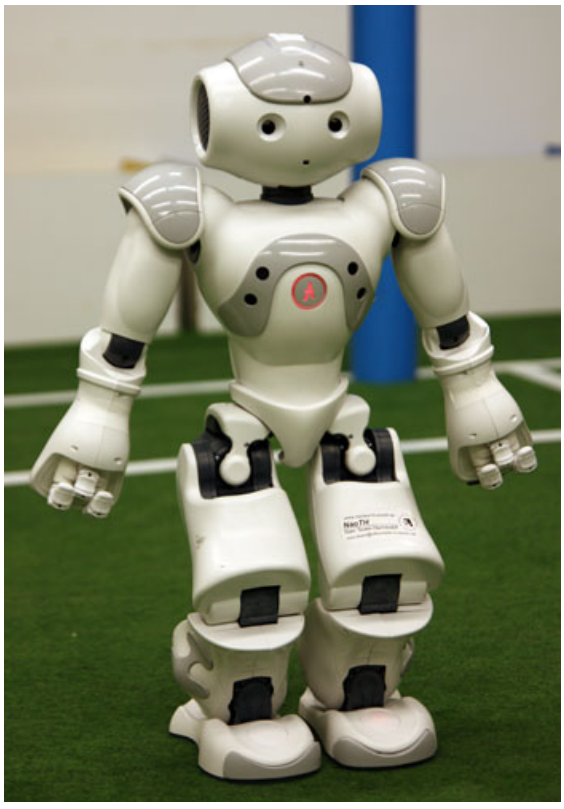
Aibo ERS-7	
Auflösung	208 × 160
Bildrate	30fps
Öffnungswinkel	
horizontal	55°
vertikal	44°
System	Aperios
CPU	576MHz
RAM	64MB SDRAM
Freiheitsgrade	20
Gewicht	1,6kg

Abbildung B.1.: Aibo ERS-7 *von Sony*

Der Vorteil dieser Plattform ist die Stabilität: auf vier Beinen kann der Aibo sich schnell und stabil fortbewegen. Auf der anderen Seite bietet seine Kamera eine sehr geringe Auflösung und braucht zudem viel Licht (bei den Spielen wurden 1000 Lux verwendet). In Verbindung mit der geringen Höhe der Kamera über dem Boden von ca. 15cm führt es insbesondere zu großen Fehlern in der Bestimmung der Entfernung zu Objekten. Ein weiteres Problem dieser Plattform sind die Gelenke, (insbesondere das Nackengelenk) die mit der Zeit großes Spiel (teilweise bis zu 10°) entwickeln, was die Wahrnehmung zusätzlich erschwert.

B.2. Nao V3+

Der humaniode Roboter *Nao* (Abbildung B.2) wird von der Firma *Aldebaran Robotics* hergestellt. Seit 2008 dient dieser Roboter als Plattform in der *Standard Platform League* des RoboCup und stellt damit den direkten Nachfolger von Aibo B.1 dar.



Nao V3+	
Auflösung	640×480
Bildrate	30fps
Öffnungswinkel	
horizontal	45°
vertikal	$34,5°$
System	Embedded Linux
CPU	500MHz
	x86 AMD GEODE
RAM	256MB SDRAM
Freiheitsgrade	21
Größe	58cm
Gewicht	4.8kg

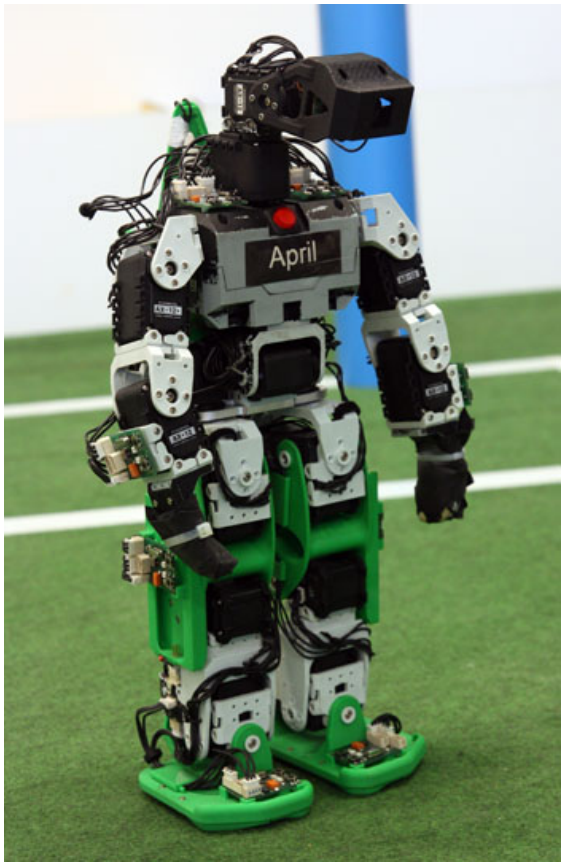
Abbildung B.2.: Nao V3+ *von Aldebaran Robotics*

Im Vergleich zum Aibo Roboter ist der Nao mit weitaus besserer Sensorik ausgestattet. Insbesondere liefert die Kamera eine viel bessere Bildqualität, was eine genauere Wahrnehmung erlaubt. Die hohe Lage der Kamera ermöglicht insbesondere eine bessere Bestimmung der Entfernung zu den Objekten. Eine besondere

Schwierigkeit stellt bei diesem Roboter die Fortbewegung dar. Das Laufen auf zwei Beinen ist weniger stabil.

B.3. A-Serie

Der humaniode Roboter der *A-Serie* (Abbildung B.3) wurde am Lehrstuhl für Künstliche Intelligenz entwickelt. Dieser Roboter basiert ursprünglich auf der kommerziellen *Biolod*-Plattform und wurde grundlegend modifiziert. Die Roboter der A-Serie wurden von dem *Humanoid Team Humboldt (HTH)* für die Teilnahme am RoboCup in der *Humanoid League (KidSize)* verwendet. Aktuell wird diese Plattform im Rahmen des Projektes *„Artifical Language Evolution on Autonomous Robots“ (ALEAR)* eingesetzt.



A-Serie	
Auflösung	320 × 240
Bildrate	5fps
Öffnungswinkel	
horizontal	$x°$
vertikal	$x°$
System	Windows CE
CPU	450MHz Intel XScale
RAM	64MB
Freiheitsgrade	21
Größe	58cm
Gewicht	4.8kg

Abbildung B.3.: A-Serie

Die Kamera des Roboters bietet die Möglichkeit verschiedene Objektive einzusetzen, wodurch der Öffnungswinkel beeinflusst werden kann. Als Computer wird ein *Fujitsu-Siemens*, *Pocket Loox* PDA eingesetzt.

Literaturverzeichnis

[1] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32, 1987.

[2] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328. IEEE, 1999.

[3] S. Enderle, M. Ritter, D. Fox, S. Sablatnög, G. K. Kraetzschmar, and G. Palm. Vision-based localization in robocup environments. In P. Stone, T. R. Balch, and G. K. Kraetzschmar, editors, *RoboCup*, volume 2019 of *Lecture Notes in Computer Science*, pages 291–296. Springer, 2000.

[4] Göhring. Cooperative object localization using line-based percept communication. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, Lecture Notes in Artificial Intelligence. Springer, 2007.

[5] D. Göhring, K. Gerasymova, and H.-D. Burkhard. Constraint based world modeling for autonomous robots. In *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2007*, 2007.

[6] D. Göhring, H. Mellmann, and H.-D. Burkhard. Constraint based belief modeling. In L. Iocchi, H. Matsubara, A. Weitzenfeld, and C. Zhou, editors, *RoboCup 2008: Robot Soccer World Cup XII*, Lecture Notes in Artificial Intelligence. Springer, 2008. to appear.

[7] D. Göhring, H. Mellmann, and H.-D. Burkhard. Constraint based localization on a humanoid robot. In *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2008*, 2008.

[8] D. Göhring, H. Mellmann, and H.-D. Burkhard. Constraint based object state modeling. In B. Herman, P. Libor, and K. Miroslav, editors, *European Robotics Symposium 2008 (EUROS)*, volume Volume 44/2008 of *Springer Tracts in Advanced Robotics*, pages 63–72, Prague, Chech Republic, 2008. Springer Berlin / Heidelberg. This volume (EUROS 2008).

[9] D. Göhring, H. Mellmann, and H.-D. Burkhard. Constraint based world modeling in mobile robotics. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2009*, pages 2538–2543, 2009.

[10] D. Göhring, H. Mellmann, K. Gerasymova, and H.-D. Burkhard. Constraint based world modeling. *Fundamenta Informaticae*, Volume 85(Number 1-4):123–137, 2008.

[11] F. Goualard and L. Granvilliers. Controlled propagation in continuous numerical constraint networks. *ACM Symposium on Applied Computing*, 2005.

[12] G. Grisetti, C. Stachniss, S. Grzonka, and Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *RSS*, Atlanta, GA, USA, 2007. Accepted for publiction.

[13] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1998.

[14] P. Heinemann, J. Haase, and A. Zell. A combined monte-carlo localization and tracking algorithm for robocup. In *IROS*, pages 1535–1540. IEEE, 2006.

[15] M. Hild, M. Juengel, and M. Spranger. Humanoid team humboldt - team description 2006. In *RoboCup 2006 - Proceedings of the International Symposium*, Lecture Notes in Artificial Intelligence. Springer, 2006.

[16] F. V. Hundelshausen, M. Schreiber, F. Wiesel, A. Liers, and R. Rojas. Matrix: A force field pattern matching method for mobile robots. Technical report, Freie Universität Berlin, 2003.

[17] L. Jaulin, M. Kieffer, Didrit, and E. Walter. *Applied Interval Analysis*. Springer Verlag, London, 2001.

[18] M. Jüngel. Bearing-only localization for mobile robots. In *Proceedings of the 2007 International Conference on Advanced Robotics (ICAR 2007), Jeju, Korea,*, August 2007.

[19] M. Jüngel. Memory-based localization. In *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2007*, 2007.

[20] M. Jüngel. Self-localization based on a short-term memory of bearings and odometry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, San Diego, October 2007. to appear.

[21] M. Jüngel and H. Mellmann. Memory-based state-estimation. *Fundamenta Informaticae*, Volume 85(Number 1-4):297–311, 2008.

[22] M. Jüngel, H. Mellmann, and M. Spranger. Improving vision-based distance measurements using reference objects. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*, volume Volume 5001/2008 of *Lecture Notes in Computer Science*, pages 89–100. Springer Berlin / Heidelberg, 2007.

[23] M. Jüngel and M. Risler. Self-localization using odometry and horizontal bearings to landmarks. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI Preproceedings*, 2007.

[24] R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

[25] M. Kieffer, L. Jaulin, Éric Walter, and D. Meizel. Robust autonomous robot localization using interval analysis. *Reliable Computing*, 6(3):337–362, August 2000.

[26] H. Mellmann, M. Jüngel, and M. Spranger. Using reference objects to improve vision-based bearing measurements. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008*, pages 3939–3945, Acropolis Convention Center, Nice, France, 22–26 Sept. 2008. IEEE.

[27] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *International Conference on Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE*, 2006.

[28] M. J. Quinlan and R. H. Middleton. Multiple model kalman filters: A localization technique for robocup soccer. In J. Baltes, M. G. Lagoudakis, T. Naruse, and S. S. Ghidary, editors, *RoboCup*, volume 5949 of *Lecture Notes in Computer Science*, pages 276–287. Springer, 2009.

[29] T. Röfer and M. Jüngel. Vision-based fast and reactive monte-carlo localization. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 856–861. IEEE, 2003.

[30] T. Röfer and M. Jüngel. Fast and robust edge-based localization in the sony four-legged robot league. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 262–273. Springer, 2004.

[31] RoboCup. Robocup official site. Available online at http://www.robocup.org, April 2010. visited on April 26 2010.

[32] T. Röfer and M. Jüngel. Vision-based fast and reactive monte-carlo localization. In *ICRA*, pages 856–861. IEEE, 2003.

[33] E. Seignez, M. Kieffer, A. Lambert, E. Walter, and T. Maurin. Experimental vehicle localization by bounded-error state estimation using interval analysis. In *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), pages 1084–1089*, 2005.

[34] A. Stroupe, M. Martin, and T. Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA-01)*, Lecture Notes in Artificial Intelligence, pages 154–165. Springer, 2001.

Liste der Algorithmen

Abbildungsverzeichnis

Index

Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Literatur und Hilfsmittel verfasst zu haben.

Ich bin damit einverstanden, dass ein Exemplar dieser Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt wird.

21. Januar 2015

Heinrich Mellmann